Development of an Object-Oriented Modeling Environment for
Prototyping Heterogeneous Simulation Models

by

Vikas Gupta

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Industrial Engineering and Operations Research

APPROVED:

[Signatures]

E. C. DeMeter, Ph.D, Chairman
M. P. Deisenroth, Ph.D, Chairman
R. S. Russell, Ph.D

December, 1990
Blacksburg, Virginia
Development of an Object-Oriented Modeling Environment for
Prototyping Heterogeneous Simulation Models

by

Vikas Gupta

E. C. DeMeter, Ph.D., Chairman
M. P. Deisenroth, Ph.D., Chairman
Industrial Engineering and Operations Research

(ABSTRACT)

Computer simulation modeling is currently the most flexible method of manufacturing system
analysis. Unfortunately, current simulation frameworks do not support the modular specifi-
cation of homogeneous and heterogeneous models.

A Simulation Program Generator (SPG) for prototyping heterogeneous simulation models is
developed. Objects in the model represent elements within a flexible assembly system.
These elements are the robot, the conveyor, the part, the schedule plan and the robot pro-
gram. These objects were modeled using the GIBSS simulation framework.

A model base and an user interface is developed to allow the construction and execution of
simulation models. Instances of the objects are created and stored into the model base.
These are retrieved later to construct a model. The user interface is provided with an exten-
sive set of tools for model creation and execution. Icons, representing objects, are selected
and placed on the terminal screen and then connected together by interaction lines to create
the complete model. The attributes of the objects can be altered and viewed on their re-
spective panes.

The SPG is a step forward in the development of computer-aided manufacturing system de-
sign environments for prototyping heterogeneous models. It allows the rapid generation and
execution of simulation models.
I wish to express my sincere appreciation to my advisor Dr. E. C. DeMeter and my co-advisor Dr. M. P. Diesenroth for their encouragement and support during the course of my research.

I am deeply indebted to Dr. DeMeter, for providing guidance and direction throughout this research. His willingness to experiment with new ideas was invaluable, especially in developing the conceptual models. I wish to thank Dr. R. S. Russell for her contribution as a member of my committee. Despite their busy schedules, each found time to review my work and to meet with me as needed. It was a pleasure and a privilege to work with them.

I wish to thank my girlfriend, Amruta, for sharing my joys and frustrations and providing personal encouragement and support, both when things looked bright and when they did not.

A special note of appreciation for LaVonda Matherly for helping in preparing this document.

Finally, I wish to thank my parents for their encouragement and guidance throughout my life. Without their love and support and their faith in my capabilities, this would not have been possible.
# Table of Contents

1 INTRODUCTION ........................................................................................................ 1
  1.1 Problem Statement ............................................................................................... 4
  1.2 Research Objective ............................................................................................. 8
  1.3 Significance of the Research .............................................................................. 9

2 LITERATURE REVIEW .............................................................................................. 10

3 METHODOLOGY ........................................................................................................ 19
  3.1 Model-Base Development ............................................................................... 20
  3.2 User Interface Development ............................................................................ 39
    3.2.1 Model Construction .................................................................................. 39
    3.2.2 Simulation Control ................................................................................... 42
    3.2.3 Data Analysis ........................................................................................... 42
  3.3 Hardware and Software Requirements ............................................................ 43
    3.3.1 Object-Oriented Programming .................................................................. 44
    3.3.2 Smalltalk Environment ............................................................................. 45
  3.4 Conclusions ........................................................................................................ 46
4 SYSTEM DESIGN .......................................................... 47

4.1 Development of Model Base ............................................. 48
  4.1.1 Class Structure .................................................. 48
  4.1.2 Simulation Mechanisms .......................................... 50
  4.1.3 Object Classes .................................................. 50
    4.1.3.1 Simulation Class Object .................................. 50
  4.1.3.2 Active Entity Objects ....................................... 53
  4.1.3.3 Passive Entity Object ..................................... 58
  4.1.3.4 Interaction Objects ....................................... 60
  4.1.4 Data Structures ................................................. 64
  4.1.5 Development of Objects ....................................... 69

4.2 Model Management .................................................. 84
  4.2.1 User Interface Development .................................... 84
    4.2.1.1 Creation of Icons ....................................... 84
    4.2.1.2 Creation of Panes ....................................... 85
    4.2.1.3 Creation of Menus ....................................... 88
  4.2.1.4 Animation .................................................. 91
  4.2.2 Model Construction ............................................... 92
  4.2.3 Executing the Model .......................................... 114
  4.2.4 Simulation Control ............................................ 115
  4.2.5 Data Analysis ................................................ 116

4.3 Conclusions ....................................................... 123

5 DEMONSTRATION ..................................................... 124

5.1 Hypothetical Assembly System ....................................... 124
  5.1.1 Model Description ............................................. 125
  5.1.2 Model Construction ........................................... 128
    5.1.2.1 Selection of Objects .................................... 128

Table of Contents
5.1.2.2 Connection of Objects ........................................... 129
5.1.2.3 Specification of Object Attributes ................................. 123
5.1.3 Execution and Control .................................................. 130

6 SUMMARY AND RECOMMENDATIONS ...................................... 139
6.1 Summary ................................................................. 139
6.2 Recommendations ....................................................... 140

BIBLIOGRAPHY ............................................................. 142

Vita .............................................................................. 144
# List of Illustrations

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.</td>
<td>Object-Oriented SPG</td>
<td>2</td>
</tr>
<tr>
<td>Figure 2.</td>
<td>A Robotic Workcell</td>
<td>5</td>
</tr>
<tr>
<td>Figure 3.</td>
<td>Heterogeneous Model of Robotic Workcell</td>
<td>7</td>
</tr>
<tr>
<td>Figure 4.</td>
<td>DEVS - Atomic Objects</td>
<td>11</td>
</tr>
<tr>
<td>Figure 5.</td>
<td>DEVS - Coupled Objects</td>
<td>13</td>
</tr>
<tr>
<td>Figure 6.</td>
<td>GIBSS - Active Entity Objects</td>
<td>15</td>
</tr>
<tr>
<td>Figure 7.</td>
<td>GIBSS - Passive Entity Objects</td>
<td>16</td>
</tr>
<tr>
<td>Figure 8.</td>
<td>GIBSS - Interaction Objects</td>
<td>17</td>
</tr>
<tr>
<td>Figure 9.</td>
<td>Hypothetical Assembly System - A</td>
<td>22</td>
</tr>
<tr>
<td>Figure 10.</td>
<td>Hypothetical Assembly System - B</td>
<td>23</td>
</tr>
<tr>
<td>Figure 11.</td>
<td>Cartesian Robot</td>
<td>26</td>
</tr>
<tr>
<td>Figure 12.</td>
<td>Uni-directional Conveyor</td>
<td>27</td>
</tr>
<tr>
<td>Figure 13.</td>
<td>Abstraction Levels of Active Entity Objects</td>
<td>30</td>
</tr>
<tr>
<td>Figure 14.</td>
<td>Abstraction Levels of Interaction Objects</td>
<td>35</td>
</tr>
<tr>
<td>Figure 15.</td>
<td>Entity and Interaction Icons</td>
<td>41</td>
</tr>
<tr>
<td>Figure 16.</td>
<td>Class Hierarchy Structure</td>
<td>49</td>
</tr>
<tr>
<td>Figure 17.</td>
<td>Simulation Class Object Logic Flow</td>
<td>52</td>
</tr>
<tr>
<td>Figure 18.</td>
<td>Active Entity Object Logic Flow</td>
<td>55</td>
</tr>
<tr>
<td>Figure 19.</td>
<td>Passive Entity Object Logic Flow</td>
<td>59</td>
</tr>
<tr>
<td>Figure 20.</td>
<td>Part Enter Object Logic Flow</td>
<td>62</td>
</tr>
<tr>
<td>Figure 21.</td>
<td>Cube Definition</td>
<td>72</td>
</tr>
</tbody>
</table>
Figure 22. Cube Data File ......................................................... 73
Figure 23. Robot Program ....................................................... 79
Figure 24. Simulation Menu ...................................................... 90
Figure 25. System Menu .......................................................... 94
Figure 26. SPG Screen ............................................................. 95
Figure 27. Part Menu ............................................................... 97
Figure 28. Abstraction Level Menu .......................................... 98
Figure 29. Conveyor Menus ...................................................... 100
Figure 30. Enter Menu ............................................................. 101
Figure 31. Part Definition Menu ................................................ 102
Figure 32. Prompters ............................................................... 103
Figure 33. Arrival Probability Menu .......................................... 104
Figure 34. Schedule Plan .......................................................... 106
Figure 35. PartForm Menu ........................................................ 107
Figure 36. Icon and Bit Editor ................................................... 108
Figure 37. Attached Icons ........................................................ 110
Figure 38. Saving Instances ...................................................... 113
Figure 39. Link Data Viewing .................................................... 118
Figure 40. Statistics Panes ....................................................... 119
Figure 41. Part Flow Chart ....................................................... 126
Figure 42. Hypothetical Assembly Model .................................... 127
Figure 43. Robot Program 1 ...................................................... 135
Figure 44. Robot Program 2 ...................................................... 136
Figure 45. Robot Program 3 ...................................................... 137
List of Tables

Table 1. Factors to be Evaluated ........................................... 6
Table 2. Factors to be Evaluated ........................................... 24
Table 3. Schedule Plan of Parts ........................................... 25
Table 4. List of Entity and Interaction Objects ....................... 29
Table 5. Attributes of Robot ................................................ 31
Table 6. Attributes of Conveyor ........................................... 32
Table 7. PartList DataBase ................................................... 66
Table 8. Data Generated of Enter and Exit Interaction Object ...... 120
Table 9. Data Generated of Conveyor Object ......................... 121
Table 10. Data Generated of Robot Object ................................ 122
Table 11. PartA Data ......................................................... 131
Table 12. PartB Data ......................................................... 132
Table 13. PartC Data ......................................................... 133
Table 14. PartD Data ......................................................... 134
Currently, computer simulation is the most extensively used systems analysis technique applied to manufacturing system design. Unfortunately, the creation and maintenance of simulation models is difficult, labor intensive and error prone. According to Balci and Nance [5], this process typically represents 70-90 percent of the total cost of a simulation analysis.

A solution to this problem is the use of an object-oriented simulation program generator (SPG). An object-oriented SPG is an interactive software tool that translates a model described by the logical assemblage of objects into the code of a simulation language, general purpose language, or machine language. A simulation object is a modeling construct which is similar in abstraction to the entity which it represents, and which hides the mechanics of its operation from a modeler.

An object-oriented SPG (Figure 1) is composed of two major software modules: a user interface and a code generator. The user interface acquires a model specification by permitting users to select, instantiate, and logically connect objects to represent major system elements. With regard to a user interface, a specification language framework dictates the types of objects offered, their levels of abstraction, and their methods of interaction.
Produces code by determining object references and dependencies, instantiating objects from a model base, and linking them into an executable model.

Specifies model through the selection, instantiation, and linking of objects.

---

Figure 1. Object-Oriented SPG
The code generator translates a model specification into either source code or object code. While methods of translation vary, a popular method used for commercial and prototype SPGs is object-oriented translation. Using this scheme, a generator creates object code by scanning a specification for object references and the identification of object dependencies. Using this information, it instantiates objects from a model base and links them together to form an executable model. With regard to object-oriented translation, a logic framework dictates how objects are combined, execute, and interact.

With respect to manufacturing system analysis, research and commercial development of object-oriented SPGs has concentrated on two applications: 1) the queuing analysis of part flow between manufacturing resources, and 2) the cycle time and geometric analysis of robotic workcells. Examples of the former include XCELL, MAST, AutoMod, and SIMFACTORY. These SPGs provide objects to represent manufacturing system entities such as resources, queues, transporters, process plans, and production schedules. Upon acquiring specifications, they create executable, discrete event simulation models. During model execution, they automatically collect data for statistics such as mean resource idle time and mean part flow time, and if requested, provide graphical animation.

Examples of the latter include IGRIP, Robot-Sim, and ROBOTICS. These SPGs provide objects to represent the geometric, kinematic, and control attributes of robots and machine tools and the kinematic and geometric attributes of fixtures and parts. Upon acquiring specifications, they generate executable models which utilize continuous simulation as well as surface modeling. During model execution, they provide a 3-D graphical animation, identify collisions between entities, and automatically collect data on robot cycle times.
1.1 Problem Statement

While excellent for the rapid specification of models at single levels of abstraction, the SPGs just described do not support the specification or generation of heterogeneous models. A heterogeneous simulation model is "one which uses multiple levels of abstraction for the representation of entities within a system" [9]. While not extensively used for the analysis of existing manufacturing systems, heterogeneous models are used during their initial, multi-level design. In particular, they are used for the collective evaluation of heterogeneously defined subsystem designs [9].

For example, consider the multi-level design of a robotic workcell (Figure 2) which consists of a robot, a conveyor, and two staging areas. A simulation model of a manufacturing system is considered multi-level if objects of equivalent type are not modeled at the same level of detail. Assume that at any given time, the kinematics and geometry of the robot and staging areas have been defined while the conveyor is specified with respect to its processing tasks and estimated time delays. Further assume that before this design is allowed to progress, that the effects of the robot instruction set on the factors listed in Table 1 need to be evaluated. To perform this analysis, a heterogeneous model such as the one illustrated in Figure 3 is required.

Objects in this model use the attributes and modeling techniques commonly applied to their respective levels of abstraction. For example, objects representing the robot and pallets manipulate kinematic and geometric attributes with continuous simulation and geometric modeling techniques. The object representing the conveyor manipulates time-delay attributes with discrete event simulation techniques. Objects representing the parts are unique in that they utilize two attribute sets: a kinematic/geometric set for interacting with the robot and staging areas, and a symbolic set for interacting with the conveyor. Particularly intriguing is
Figure 2. A Robotic Workcell
HETEROGENEOUS SIMULATION OBJECTIVES

To provide data on:

1. Part position
2. Part velocity
3. Conveyor velocity
4. Position of parts on conveyor
5. Conveyor working times
6. Robot idle times
7. Robot working times
8. Robot joint positions
9. Robot joint velocities
10. Robot link positions
11. Robot link velocities

Table 1. Factors to be Evaluated.
Figure 3. Heterogeneous Model of Robotic Workcell.
the modification of these attributes during the simultaneous interaction of part, robot, and conveyor objects.

1.2 Research Objective

The objective of this research is to prototype an object-oriented simulation program generator capable of supporting the modular specification and generation of heterogeneous models. The intent of this research is to test the feasibility of such an environment rather than the development a commercial grade product. Thus the proposed SPG will offer only a limited selection of objects and will not generate models optimized with respect to efficiency.

In general, the proposed SPG will provide objects which represent entities within a hypothetical flexible assembly system. The elements of the assembly system represented will the robot, the conveyor, the part, the schedule plan and the robot program. The objects will be at differing levels of abstraction. To demonstrate its feasibility, the SPG will permit:

1. the specification of homogeneous or heterogeneous models of the assembly cell through the selection and logical assemblage of objects from its model base,
2. the conversion of a specification into an executable model via object-oriented translation, and
3. the interactive control and graphical animation of an executing model.
1.3 Significance of the Research

The creation of the proposed SPG will be an important step toward the development of a new generation of computer-aided manufacturing system design environments. The use of these environments will drastically reduce the labor time and costs associated with design and computer modeling during the multi-level design of manufacturing systems. The development of the SPG will result in a better design of manufacturing subsystems with less changes in the initial stage. Using these environments, engineers will be able to design manufacturing subsystems, develop models of these designs, store these models in a model base, and quickly assemble these models whenever the designs are evaluated collectively.
Crucial to the successful implementation of the proposed SPG is the use of the proper language and logic frameworks. This section discusses object-oriented SPG frameworks that have been reported in the literature.

Zeigler [21] was one of the first to formalize the model generation process with his development of the DEVS (Discrete EEvent Simulation) language and logic frameworks. Zeigler’s intent was to enhance the modular, top-to-bottom specification and bottom-up implementation of discrete event models. The DEVS language framework classifies objects as either atomic or coupled. When used for model specification, these objects are manipulated in a fashion similar to that of subroutines and subordinate subroutines during traditional software development.

Atomic objects (Figure 4) are the fundamental units of model specification. They represent subsystems, which utilize sets of input/output ports to communicate with their environment. During simulation, atomic objects change their internal states in response to the passage of time or the arrival of messages. By transmitting messages, atomic objects not only represent subsystems, but their interactions as well.
Figure 4. DEVS - Atomic Objects
Coupled objects (Figure 5) control assemblages of interacting atomic objects and serve to replace them within model specifications. Coupled objects utilize input/output ports in the same fashion as atomic objects. However, instead of simulating subsystem behavior directly, they transmit messages received at their input ports to the input ports of their subordinate atomic objects. Likewise, they transmit messages from the output ports of their subordinate atomic objects via their output ports.

The DEVS logic framework is based explicitly on the object-oriented programming paradigm. Coupled objects and atomic objects are coded with hierarchical object classes. Each class utilizes the methods and data structures required to simulate some abstraction of functional or logical behavior. In addition, each class works in combination with a processor object class to execute synchronously within a composite model.

Processor object classes include simulators, co-ordinators, and root-co-ordinators. Simulators synchronize the execution of atomic objects, while coordinators synchronize the execution of coupled objects. Root-coordinators synchronize the execution of simulators and coordinators within composite models, manage system clocks, and link directly to coordinators at the top of the object hierarchy. During simulation execution, messages are passed between processor objects. In turn, processors interpret these messages and simulate subsystem behavior by passing messages to their respective atomic or coupled objects.

Antonelli, Volz, and Mudge [1] developed the PDL (Procedural Decomposition Language) frameworks. Their intent was to enhance the modular top-to-bottom specification of manufacturing system models. Using the PDL language framework, objects are classified in a fashion similar to atomic and coupled objects within the DEVS frameworks. Namely, they represent subsystems which change their internal states with respect to time or the arrival of messages. The PDL logic framework is similar to DEVS, with the exception that its is designed around ADA programming constructs and the use of multiple processes.
Figure 5. DEVS - Coupled Objects
DeMeter [9] developed the GIBSS (Generalized Interaction Based Simulation Specification) framework to support simulation modeling during the multi-stage design of manufacturing systems. The GIBSS language framework was designed to enhance the modular specification of heterogeneous models. It classifies objects as active entity, passive entity or interaction.

Active entity objects (Figure 6) represent system entities whose behavior is relevant enough to simulate on an individual basis. Active entity objects change their internal state in response to the passage of time or stimuli from their environment. Unlike, atomic objects, stimuli are not introduced via messages, but rather by the direct manipulation of attributes by interaction objects.

Passive entity objects (Figure 7) represent system entities whose attributes are relevant to a simulation, but whose behavior is not relevant enough to simulate on an individual basis. A passive entity object has the characteristic of being able to change the number and types of attribute sets it possesses. Each attribute set represents a different level of abstraction. Manipulations by interaction objects ultimately dictate which attribute sets are used.

Interaction objects (Figure 8) represent the interactions between entities. To simulate interactions, they monitor the passage of time or changes in the internal states of entity objects. If the conditions of an interaction are met, they simulate the results of the interaction by modifying the attributes of the interacting entity objects.

The GIBSS logic framework was designed to facilitate the modular assembly of objects using heterogeneous simulation techniques. As a result, it is much broader in scope than DEVS and PDL in that it includes mechanisms from both discrete event and continuous simulation. In addition, it is activity oriented rather than event oriented.

An executable GIBSS model is comprised of instantiated classes of active entity, passive entity, and interaction objects bound together via a master simulation object. Active entity ob-
Figure 6. GIBSS - Active Entity Objects
Interaction Object 1
Input/Output

STATIC PASSIVE ENTITY OBJECT

Attribute Set 1
Attribute Set 2
Attribute Set 3

Interaction Buffer
Dynamic Attribute Logic

Interaction Object 2
Input/Output

Interaction Object 3
Input/Output

Request for Attribute Values From Rel. Data Base

Attribute Values from Rel. Data Base

Figure 7. GIBSS - Passive Entity Objects
Figure 8. GIBSS - Interaction Objects
jects and interaction objects execute logic flows which permit them to execute combinations of time-based activities, conditional activities, and time-based attribute functions. When combined with the logic flow of a master simulation object, the composite logic flow resembles the three-phase approach modified to handle the execution of timed-based attribute functions. Passive entity objects execute a logic flow which enables them to manage and simultaneously update multiple attribute sets. To do so, they utilize a relational database which links attributes and their associated values.

Thomasma and Ulgen [18] developed an object-oriented SPG (Smart-Sim) for the hierarchical modeling of manufacturing workcells. Their system utilizes the DEVS frameworks in combination with the SmallTalk-80 software environment. To specify discrete event models, users select and manipulate icons which represent atomic models of machines, tools, robots, and conveyors. To indicate material and information flow, users graphically connect icons with directed arcs. To create coupled objects, users place windows around groups of icons and subsequently save the image. These coupled objects are then capable of the same manipulations as atomic objects.

Zeigler [22] developed an object-oriented SPG (DEVS-Scheme) which supports the development of atomic and coupled objects. Rather than application specific, Zeigler's system permits users to create their own atomic and coupled model classes. In addition, users may instantiate classes and store model data into a model base. From this model base, users may select atomic and coupled objects via icons, arrange these icons on a video screen, and logically connect these objects to create a composite model.
3 METHODOLOGY

To successfully implement the proposed SPG, a suitable language framework and logic framework was required. GIBSS was chosen because it was specifically designed for the modular specification and execution of heterogenous models. In particular, GIBSS provides:

1. Active entity objects which simulate the functional, logical and/or physical behavior of major system entities,

2. Passive entity objects which represent minor system entities and which dynamically alter their attributes to meet the requirement of their interactions, and

3. Interaction objects which simulate the interactions between manufacturing system entities and which bind heterogeneous and homogeneous entity objects together.

In general, it was felt that the Simulation Program Generator created in this research should:

1. provide a model-base containing GIBSS classified objects representing entities and interactions within an hypothetical assembly system,

2. permit users to specify homogeneous and heterogeneous models with objects from the model base through the selection and manipulation of icons,

3. accept an iconic specification and execute the model, and
4. permit users to interactively control the execution of a model as well as provide an animated display of its execution.

The development of the SPG involved:

1. the creation of a model base, and
2. the creation of user interface.

The development of these steps is discussed next.

3.1 Model-Base Development

For a better understanding of the development of the Model Base, an hypothetical flexible assembly system is discussed. The system is depicted in Figure 9. Four types of parts enter the system, PartA, PartB, PartC and PartD. Their schedule plans are shown in Table 2. The system consists of three robots and three conveyors. For the sake of simplicity, conveyor 1 is divided into three sections and is depicted in Figure 10. Further, attributes of the elements in the system that are known are listed in Table 3. These attributes are considered to determine the abstraction level of the entities.

Parts enter the system at conveyor 1a and travel to robot 1. At robot 1, an assembly operation is performed according to its robot program. After the assembly operation is over, the product travels down either conveyor 2 or conveyor 1b. At the end of conveyor 1b, the product has the option to travel down conveyor 3 or have an assembly operation performed by robot 2. The decision is based on the schedule plan of the part. At the end of conveyor 3, the product is removed from the system. If an assembly operation is performed by robot 2, then the product
travels down conveyor 1c to robot 3. At robot 3 another assembly operation is performed and
the product exits the system.

The first step in the development of the model base of the SPG involved the identification of
the elements of the hypothetical flexible assembly system being simulated. The system was
divided into three assembly cells and a material handling cell. The material handling cell was
represented by two conveyors and the assembly cell by three robots. Further, the part en-
tering into the system, their schedule plan and the robot program were also designed. Each
of these elements are discussed with respect to their functional properties.

3.1.1 System Elements

Robot: The hypothetical robots designed in the system are Cartesian robots (Figure 11) hav-
ing six links. The link dimensions are given in Appendix A. All dimensions of the robot are
fixed and cannot be changed interactively within the SPG. The movement of these links are
controlled with a robot program which specifies their position and velocity.

Conveyor: A uni-directional conveyor was designed which had a maximum capacity of ten
parts. The conveyor is shown in Figure 12 and the dimensions of its links are shown in Ap-
pendix A. The conveyor can be divided into sections, as in the case of the hypothetical as-
sembly system being studied. The conveyor was designed such that its velocity could be
changed interactively during simulation execution. Note that to simplify the kinematic and
geometric modeling of the robot, conveyor, and parts, all motion within the workcell are as-
sumed to be linear, and all geometry are considered to be cubic.

Schedule Plan: The robot is designed to perform an assembly operation depending upon the
schedule plan of the part. The schedule plan contains the route to be taken by a part to form
a complete product. The identification of the robots, to which the part is to be routed to, is
Figure 9. Hypothetical Assembly System - A
Figure 10. Hypothetical Assembly System - B
<table>
<thead>
<tr>
<th></th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Conveyor 1 &amp; 3</strong></td>
<td>Travel Time</td>
</tr>
<tr>
<td><strong>Conveyor 2</strong></td>
<td>Capacity</td>
</tr>
<tr>
<td></td>
<td>Travel Time</td>
</tr>
<tr>
<td></td>
<td>Velocity of conveyor</td>
</tr>
<tr>
<td><strong>Robot 1</strong></td>
<td>Execution time</td>
</tr>
<tr>
<td><strong>Robot 2</strong></td>
<td>Execution time</td>
</tr>
<tr>
<td></td>
<td>Joint velocities</td>
</tr>
<tr>
<td><strong>Robot 3</strong></td>
<td>Execution time</td>
</tr>
<tr>
<td></td>
<td>Joint velocities</td>
</tr>
<tr>
<td></td>
<td>Link velocities</td>
</tr>
</tbody>
</table>

Table 2. Factors to be Evaluated.
<table>
<thead>
<tr>
<th>Part</th>
<th>Actions</th>
</tr>
</thead>
</table>
| PartA | robot 1 add Part1  
|       | robot 2 add Part2 |
| PartB | robot 2 add Part2  
|       | robot 3 add Part3 |
| PartC | robot 1 add Part1  
|       | robot 2 add Part2  
|       | robot 3 add Part3 |
| PartD | robot 3 add Part3  
|       | robot 4 add Part4 |

Table 3. Schedule Plan of Parts
Figure 12. Cartesian Robot
Figure 13. Uni-directional Conveyor
stored in the schedule plan. The schedule plan is checked before a part is allowed to go to a robot. If the robot identification number matches the first entry in the schedule plan, then the part is routed to the robot otherwise, it exits the system. At each robot in the simulation, there is a part entering into the system, which is to be assembled to the product. The identity of this part is also stored in the schedule plan.

**Robot Program:** Each robot executes a simple program which has a set of commands for performing the assembly operation. This robot program was designed to be as simple as possible for a user to create. It contains a set of three commands which describe the movement of the links of the robot. A PMOVE command moves the joints and the links and calculates their position and velocity. The DELAY command stops the movement of the joints and the links for a specified period of time. The WRiTEP command sends a signal to introduce a new part at the robot. The robot program was not based on any existing robot programming language.

### 3.1.2 Model Requirements

Model-base objects were classified using the GIBSS language and logic frameworks. Entities within the cells were modeled at three levels of abstraction: 1. functional/symbolic, 2. kinematic/symbolic, and 3. kinematic/geometric. Entity objects and interaction objects which appear in the model-base are listed in Table 4. The levels of abstraction to which these objects are modeled are shown in Figure 13 and Figure 14.

Model-base objects were modeled by taking their attributes and operations into consideration. The attributes of the objects are listed in Tables 5 and 6. The attributes and operations of the objects considered are based on the simulation model to be created. Attributes that are similar for all the objects were grouped together into a higher hierarchical class. The objects could then operate on these attributes by referring to that class.
Active Entity Objects

Robot
Conveyor

Passive Entity Objects

Part

Interaction Objects

Part Arrival
Part Departure
Transfer
Assemble
Collision Check

Table 3. List of Entity and Interaction Objects
Figure 14. Abstraction Levels of Active Entity Objects
1. Capacity
2. Travel Time
3. Number of parts received
4. Number of parts sent
5. Accepting/notAccepting
6. Icon
7. Previous object
8. Next object
9. Position of parts
10. Velocity

Table 6. Attributes of Conveyor

Symbolic Conveyor

Kin/Geo Conveyor
The operation to be performed by each object depended upon the behavior of the object and its functional requirement. For example, the gripper of the robot was to be simulated to grasp a part. The operations required of the gripper was to check its position, check the part position, check for collision detection, update the position of the part and itself and change its velocity. The identification of the operations performed by each object and their functional requirements helped to make each object modular and without their depending upon any other specific object.

The functional requirements of the entity objects at different levels of detail are discussed next.

**Symbolic Conveyor:** A Symbolic Conveyor accepts a part when its current capacity is less than its maximum capacity. Upon accepting a part, it schedules the departure time of the latter. The positions of the parts while they are on the conveyor are unknown. In the case of the hypothetical assembly system modeled in the previous section, conveyor 1a, 1b and 1c are Symbolic Conveyors since their travel time is the only attribute that is known.

**Kin/Geo Conveyor:** A Kin/Geo Conveyor uses continuous simulation to simulate the movement to the parts on it. The position and velocity of each part on the conveyor is known. The velocity of the part is assumed to be equal to that of the conveyor. Since the position and the velocity of the parts and the conveyor are known, therefore conveyor 2 can be modeled as a Kin/Geo Conveyor.

**Symbolic Robot:** A Symbolic Robot accepts the part when it is in the idle state. Upon accepting the part, it schedules the departure time of the latter by calculating the time taken to execute its robot program. Since the only attribute known of robot 1 is the execution time therefore it can only be modeled as a Symbolic Robot.
**Kinematic Robot:** A Kinematic Robot executes its robot program and simulates the movement of its joints. The joint velocities and positions are calculated according to the data provided by the robot program. After the entire program has been executed, the product is allowed to depart from the robot. Robot 2 has attributes of joint velocity and position and therefore it can be modeled as a Kinematic Robot.

**Geometric Robot:** In addition, to the simulation of the joints, a Geometric Robot also simulates its links. The velocity and positions of the links are calculated after the execution of each command of the robot program. Also the position of the part or product is updated. In the case of robot 3, the link dimension and position are known therefore it can be modeled as a Geometric Robot.

To facilitate the construction and execution of a heterogenous model, a set of interaction objects were developed. These interaction object were designed to simulate a particular interaction, using attributes which are particular to the entity objects involved. The interaction objects along with their levels of abstraction are shown in Figure 14. The function of each of these interaction objects is discussed next.

**Part Enter:** Parts arrive into the system at a Part Enter object. The object provides each part with a dynamic data structure which contains the schedule plan of the assembly operation. The arrival probability and the type of part arriving is determined by this object. The Part Enter object will send the part to the next object, if and only if, it is possible for the latter to accept it, otherwise it is thrown out of the system.

**Part Exit:** A Part Exit object works as a sink which accepts all types of parts arriving to it. This object removes parts or products from the system.
Figure 14. Abstraction Levels of Interaction Objects
Collision Check: A Collision Check object is used for determining a geometric intersection between part and the robot gripper or the conveyor bed. The object replies in the affirmative or negative to a collision check request.

Assemble: An Assemble object orders a robot to perform the assembly operation upon receiving a product from the transfer system.

Transfer: A Transfer object serves as a router which determines the object to which a product should go.

3.1.3 Development of Objects

The major steps taken in the development of the objects were:

1. the creation of classes,
2. the design of the Part data structure,
3. the design of the logic flow, and
4. the creation of the methods.

3.1.3.1 Creation of Classes

Two classes were created which were subclasses of the Smalltalk Object Class. These were the Simulation Class and the SimulationObjects Class. The SimulationObjects Class had subclasses which represented the entity and interaction objects in the simulation. Further, each active entity class had subclasses representing their levels of abstraction. These classes were created using the Class Hierarchy Browser of Smalltalk.

The Simulation Class controlled the simulation and maintained the simulation clock and the queue of events. The SimulationObjects Class scheduled the execution of each object.
3.1.3.2 Design of Part Data Structure

The part entering into the system has a dynamic attribute structure represented by the PartList database. During simulation execution, the part object changes the number and types of attribute sets that it possesses in accordance with its interactions. Each attribute set represents a different level abstraction. When assigning a new attribute value, the part object cross references attributes, by using the PartList database, whenever it possesses multiple attribute sets. During a cross reference, an object retrieves attribute values which directly corresponds to the newly assigned value. It then assigns these values to the corresponding attributes within other sets.

The attribute set of the part is unique. Consider the case of the hypothetical assembly system discussed in Section 3.1. The part has symbolic attributes while with conveyor 1c and kinematic/geometric attribute when with robot 2.

3.1.3.3 Design of Logic Flow

The Simulation Class logic flow started with the initialization of its variables. The simulation clock was then updated and the time-based events and attribute functions at the current time were executed. After this phase is over, the conditional scan phase is executed in which all activities with beginning and ending time equal to the current time are executed. After all the events at the current time are executed the class checks whether interaction is required. If it is, then the simulation enters the model control phase. After this phase is over, the clock is updated and the logic flow is repeated until there are no more events to execute.

The logic flow of the active entity objects starts with the initialization of their variables and data structures. After the initialization phase is over, the event list is checked whether there are any events of the entity object scheduled for the current time. If they are, then the time-based events and attribute functions are executed. After this phase is over, the conditional
scan is executed in which all conditional activities occurring at the current time are fired. After the completion of this phase, the next event is scheduled and the logic flow is repeated.

The passive entity utilizes a logic flow which simply manipulate its attribute sets. The logic flow allows the creation or destruction of attribute sets and finds or changes the values of attributes.

The logic flow of the interaction objects is similar to that of the active entity objects.

3.1.3.4 Creation of Methods

The methods of each object were created so that they were inherently reusable by all objects within the class. The methods created were independent and could be executed by sending a message to them. Methods that were common to all entity objects were placed in a superclass. These methods could still be executed from each entity class. For example, the animation methods of each entity object were similar so they were placed in the SimulationObjects Class, which was the super class of all entity objects. With the help of all the methods, the attributes of the entities were calculated and updated.

The implementation of the objects involved the execution of the methods by sending and receiving messages to and from it. Since each object created was modular in nature, therefore only attributes pertaining to that object were calculated and altered without affecting other objects.

In summary, the steps taken in the development of the model-base were:

1. design of the hierarchical class structure.
2. creation of the data structures for storing the model and attributes of the entities.
3. design of the kinematics, geometry, and control architecture of the assembly and material handling cells,

4. coding of the master simulation object,

5. coding of the active entity objects at the three levels of abstraction,

6. coding of the passive entity objects, and

7. coding of the interaction objects for the three levels of abstraction.

3.2 User Interface Development

A user interface module was developed to allow a user to:

1. construct assembly cell models through the graphical manipulation of icons and windows,

2. interactively control simulation execution, and

3. examine simulation execution via graphical animation and observe attribute values associated with entity objects and the relational database.

Each of these functions are discussed next.

3.2.1 Model Construction

The creation of the user interface required the development of an user friendly, interactive modeling environment. An extensive set of icons, menus and windows were created for this purpose using the classes provided in Smalltalk. Icons and their associated windows were created for each entity object in the model-base. Icons for interaction modeling were created for interaction sets rather than interaction objects. An interaction set is a group of interaction objects which simulate the same event but at different levels of abstraction. For example, the
transfer interaction between a conveyor and a robot could be modeled geometrically, kinematically, or symbolically. Thus three different interaction objects could be used to model this phenomenon. When an interaction icon is logically connected to an entity icon, the user interface will select the object within the interaction set which corresponds to the level of abstraction of the entity object.

A simulation model can be created or retrieved from the model base. The model base stores instances of objects which are initialized when they are retrieved. Each instance of an object stored in the model base has an entire set of instance variables and data structures. The instance variables are provided to an object instance when it is retrieved.

During a modeling session, a user specifies a model by selecting icons, placing them in a specification area on the terminal screen, and connecting them graphically with interaction lines. Entity and interaction object icons are illustrated in Figure 15. Instances of objects are created and a chain is formed in which each object carries with it an instance of its previous object and next object. This model structure has been adopted to achieve modularity in the simulation model.

To test the functioning of the SPG on a hypothetical flexible assembly system, instances of a conveyor and a robot at different levels of abstraction, were created and stored into the model base. Further, the data structures of each object are initialized and stored with their default values. These instances were later retrieved and a model constructed of an assembly system. Also, instances of interaction objects were created and connected to the conveyor and robot objects to form a complete model.

Executable code is not generated for running the model but instances of objects are created and messages sent to their methods to begin execution. The execution of a method proceeds by sending messages to other methods and receiving their output. This output may be an integer, an ordered collection, a dictionary or an instance of a class. The execution proceeded
Figure 15. Entity and interaction icons
with the transfer of the part data structure from one instance to another. Since each object knew the identity of its next object, therefore the part flowed through the system.

The data required for an executable model depends upon the abstraction level of its constituent elements. Data for object instances are stored in files. A large number of files of the geometric and kinematic data of the robot and the conveyor were constructed. Data from these files are read and stored into data structures upon the initialization of entities.

### 3.2.2 Simulation Control

Once a model has been specified and translated, the user interface permits a user to control model execution interactively. To do so, the user interface allows a user to start and stop an executing simulation. If the execution is stopped, then the attributes of an object instance are frozen and stored. Upon restarting the execution, the attributes of an object instance acquires its previous values and the simulation starts from where it left off.

Features to single step the simulation or to set breakpoints have not been incorporated into the simulation control.

### 3.2.3 Data Analysis

The user interface permits a user to examine the attributes of entity objects within the model as well as attribute values within the part database. The simulation model provides information on the attributes of each of the executing components. Object attributes are constantly updated as a simulation proceeds. The information provided on each object depends upon its
abstraction level. Tables 5 and 6 provide a list of all the attributes of objects that are capable of being examined.

The data for each entity within a model is generated and viewed on a Statistics Pane. Each object is provided with a Statistics Pane, which is a window which displays object values. This pane allows a user to view data, but not to change it. A pane is updated whenever there was a change in attributes of the entity.

In summary, the steps taken to create the user interface were:

1. the creation of an extensive set of icons, panes and menus,
2. the creation of the model specification format,
3. the creation and coding of the animation screens,
4. the creation of the Rules of Interaction,
5. the creation and coding of the interactive control features, and
6. the coding of the data generation methods.

### 3.3 Hardware and Software Requirements

An IBM PS/2 was used as the hardware platform for SPG development. Its selection was based on its speed, availability, and support for color graphics. The SmallTalk/V environment was used as the software platform. Its selection was based on its object-oriented programming and iconic modeling facilities. To gain a better understanding of the construction of the Simulation Program Generator, an overview of object-oriented programming and the Smalltalk environment is discussed.
3.3.1 Object-Oriented Programming

Classes, methods, objects and messages form the basis of programming in the object-oriented environment. Each of these concepts is discussed next.

3.3.1.1 Classes

Classes describe data structures (objects), algorithms (methods) and external interfaces (message protocol), within an object-oriented environment. A class describes the implementation of a set of objects that represent the same type of system component. Classes form a hierarchy, consisting of a root class, called object, and many subclasses. Each class inherits the functionality of its superclasses in the hierarchy.

3.3.1.2 Objects

Objects are self-describing data structures. Every object is an instance of a class. All objects which are instances of a class are similar because they have the same structure (i.e. the same instance variables), the same messages to which they respond, and the same available methods.

Each object has a set of instance variables and class variables. The instance variables exist for the lifetime of the object while the class variables exist until they are explicitly deleted.

A large number of class variables and instance variables were defined to create the SPG. Some of the variables created are discussed in Section 4.1.4.
3.3.1.3 Messages

A message is a request for an object to carry out one of its operations. A message specifies which operation is desired, but not how that operation should be carried out. The receiver, the object to which the message was sent, determines how to carry out the requested information.

3.3.1.4 Methods

Methods are the algorithms which are performed by an object in response to receiving a message. Methods represent the internal details of the implementation of an object.

3.3.2 Smalltalk Environment

Smalltalk is a graphical, interactive programming environment. It provides a user with a built-in set of classes and methods. These classes are arranged in an hierarchical order to allow for the inheritance of methods and variables. All classes are subclasses of the Object Class. The existing classes can be altered or new ones created. The new classes created can be a subclass of the Object Class or any other class. The creation of a new class involves the specification of the class name, its superclass, variable names and a set of methods. A subclass is completely contained within its superclass, i.e., all instances and methods of the superclass are instances and methods of the subclass but, not vice versa.

The Smalltalk environment contains several types of windows. Each window can be further subdivided into panes. Each pane has a pop-up menu associated with it. The Class-Hierarchy Browser and the Debugger windows play an important role in program creation. The Class-Hierarchy Browser is used for viewing the classes and their methods and also for creating
new ones. This window has a set of six panes which allow the selection of the superclass, definition of the class, definition of the instance and class variables and the creation of methods. The Debugger allows for easy inspection and correction of errors. The user is provided with a complete set of methods executed till the error occurred. The methods can then be altered and saved and the execution of the program can be restarted from where it left off.

3.4 Conclusions

A Simulation Program Generator was created which utilized GIBSS based entities. The functional requirements of the SPG created were discussed in this chapter. In summary the creation of the SPG involved:

1. The specification of the types of simulation models created and executed on the system along with a description of the system entities and their levels of details, and
2. The design and creation of a user interface for model construction with the tools provided and the ability to store this model. The ability to control the execution of a model and the generation of data for each of the entities in the executing or executed model was also created.

The next chapter discusses the creation of the SPG in detail along the lines of its functional specifications. The demonstration of the SPG on an example model is shown in Chapter 5.
A Simulation Program Generator was designed to construct and execute modular heterogeneous models of flexible assembly systems. As described in the previous chapter, the development of the SPG involved the development of a model base and a user interface.

The model base allows the storage and retrieval of objects. Objects are stored in the model base in the form of instances along with their variables. The user interface provides an environment for the creation, execution, and graphical animation of a model.

The development of the model base and the user interface are described in this chapter.
4.1 Development of Model Base

The development of the model base involved the design of a class structure, data structures, objects and their simulation mechanism. Each of these developments are discussed next.

4.1.1 Class Structure

To create the SPG, two main classes were created as subclasses of the Object Class. These were the Simulation Class and SimulationObject Class. The SimulationObject Class further has subclasses which represent the entities and interactions in the manufacturing system. The hierarchy is depicted in Figure 16.

The Simulation Class maintains the simulation clock and the queue of time based events. The specification of the arrival of new objects into the system and the specification of resources are coordinated in this class. The purpose of Simulation Class is to manage the topology of simulation objects and to schedule actions to occur according to simulated time. Instances of Simulation Class maintain a reference to a collection of SimulationObjects, to the current simulated time, and to a queue of activities waiting to be invoked.

The SimulationObjects Class describes objects that appear in a simulated situation. A SimulationObject is any object that can be given a sequence of tasks to do. Each object defines a main sequence of events that is initiated when the objects enter the system.
Figure 16. Class Hierarchy Structure
4.1.2 Simulation Mechanisms

As stated in Chapter 3, the objects in this SPG are similar to GIBSS objects and are classified as active entity objects, passive entity objects and interaction objects. The active entity objects are the robot and the conveyor, the passive entity object is the part/product and the interaction objects are the Part Enter, Part Exit, Part Transfer, Assemble and Collision Check.

4.1.3 Object Classes

4.1.3.1 Simulation Class Object

The simulation class object is responsible for synchronizing the simulation of a composite model. The simulation class object executes a logic flow that is similar to the three phase approach and is shown in Figure 17. It synchronizes the events of the active entity objects and the interaction objects. In addition, a simulation class object is the only object which directly interacts with the user during simulation execution.

As described in Section 4.1.1, the Simulation Class controls a simulation. The Simulation Class has two primary data structures, the simTime and the eventChain. The simTime is a data structure which maintains the current time of the simulation. The eventChain is a data structure which maintains a list of time-based events waiting to be executed.

Upon starting its logic execution, the simulation class object enters the initialization phase. This phase involves:
1. the initialization and setup of the Icon Pane and the Simulation Pane,
2. the initialization of the active entity object, passive entity object and the interaction object attributes,
3. the creation and storage of object icons in the iconsSelectedTillNow dictionary,
4. the initialization of the simTime,
5. the creation and initialization of the eventChain,
6. the creation and initialization of the PartList database, and
7. the reading of geometric data for robots and conveyors and its storage into a data structure.

After the initialization phase is over, the clock update phase is executed. The future occurrence time of each active entity and interaction object is checked from the eventChain. The simulation clock simTime is updated to the lowest of the time values in the eventChain.

Subsequently, a simulation class object enters the time-based event phase, where it sequentially activates each active entity object and interaction object. Upon activation, these objects execute all time-based events with beginning and ending times equal to the value of simTime. In addition, they also execute all time-based attribute functions for the increment in simulation time. Upon completion of this phase, the conditional activity phase is entered. During this phase, each object checks their own conditional activities and executes them. After the conditional activities are executed, the simulation class object checks whether the execution termination command has been given. If not, then the object goes back to the clock update phase. If termination conditions are met then, the simulation statistics are gathered and transferred to the environment.

The simulation class object can be interrupted at any time of its execution by clicking the left button of the mouse. The execution can be restarted by selecting the execute option from the Simulation Menu.
Figure 17. Simulation Class Object Logic Flow
4.1.3.2 Active Entity Objects

The active entity objects provided by the SPG are used to represent robots and conveyors. Each active entity is further classified according to its level of abstraction. The execution of each active entity is coordinated by the master simulation object. Also, the logic flow of the entity object is similar to that of the master simulation object and is shown in Figure 18.

The logic flow starts with the initialization phase in which instance variables of the active entity's superclass are initialized. Then, the instance variables of the entity are initialized and set to their default values. Also, created and initialized are the data structures of each entity object.

After the initialization phase is over, the clock update is executed. The future occurrence time of the entity object is scheduled by storing it into the eventChain of the master simulation object. The logic flow continues with the entity object entering into the time-based event phase. Each entity checks the master simulation eventChain to determine if it has to execute. The object with the lowest time on the eventChain is sent a message to execute. The execution of each entity proceeds with a change in the values of its variables. Also executed are the time-based attribute functions. After the completion of these phases, the conditional scan phase is executed. Conditional activities are executed for the increment in simulation time. These conditional activities occur due to a change in attributes of the entity. The statistics update phase is executed after the completion of the conditional scan phase. Data on all the attributes that changed during the increment of time are updated.

This logic flow continues until there are no more events stored in the eventChain. If there are any more events, then the logic flow is executed again with the exception of the initialization phase.
A short description on each active entity object at different levels of abstraction is discussed next.

### 4.1.3.2.1 Symbolic Conveyor

The symbolic conveyor is an entity to which parts arrive at a certain time and depart after a certain time interval. The time interval is preset with the help of the Probability Menu. The conveyor instance maintains a set of five icons which represent the current capacity of the conveyor. These icons were created only for the purpose of animation. Further, statistics are maintained for the number of parts entering the conveyor and the number leaving it.

The initialization phase of the logic flow of the symbolic conveyor involves:

1. the initialization and creation of the FIFO queue.
2. the initialization of the icons of the conveyor.
3. the creation of data structures to hold the statistics of the conveyor.

The symbolic conveyor accepts the part and schedules the travel time of the latter onto the eventChain. After the elapse of the set time, the event is removed from the eventChain and the part is sent to the next object. The icon of the conveyor keeps changing with respect to its current capacity.

### 4.1.3.2.2 Kin/Geo Conveyor

The Kin/Geo Conveyor differs from the symbolic conveyor in terms of the attribute set and the method of execution. This object takes into account the geometry of the conveyor, its velocity and the position of products on it. Further, the kin/geo conveyor uses continuous simulation while, the symbolic conveyor uses discrete simulation.
Figure 18. Active Entity Object Logic Flow
The simulation of the kin/geo conveyor starts off with the initialization phase which involves:

1. the initialization of the variables and data structures described previously,
2. the initialization of the velocity of the conveyor,
3. the creation of a PartPosOnConv dictionary which stores the positions of each part on the conveyor, and
4. reading of the data files containing the geometry of the conveyor bed and legs and storing into a data structure.

The acceptance of the part by the kin/geo conveyor is similar that of the symbolic conveyor. The acceptance of the part is a conditional activity and is executed only when the part is ready to depart from one object and be accepted by the other. After the part is accepted by the conveyor, its icon changes and the part is placed at the beginning of the conveyor. The movement of the part is then scheduled for the current time. After the elapse of a certain time interval all the parts on the conveyor are moved forward. The distance moved is dependent upon the velocity of the conveyor. After every move, the conveyor is scheduled for another move for a set time interval. This process continues till a part reaches the end of the conveyor. The part is then withdrawn and sent to the next object.

4.1.3.2.3 Symbolic Robot

The working of a Symbolic Robot is dependent upon its robot program and the presence of an Assemble interaction object. If an Assemble interaction object is not attached to the robot, then a part will be accepted by the robot, but scheduled to be sent out immediately without any processing.

During the initialization phase the initialization of the state of the robot and the statistics on the working time and idle time of the robot takes place.

4 SYSTEM DESIGN
The processing time of a robot is determined by its robot program. During execution, a robot program is scanned and the time taken to execute the entire program is set as the departure time of a part from the robot. Upon acceptance of the part, the state of the robot changes from idle to working and a corresponding change is reflected in the icon shape. When a part departs the entity, the robot is set back to its initial state.

4.1.3.2.4 Kin/Geo Robot

The Kin/Geo Robot takes into account the geometry and kinematics of the links and joints of the robot. The positions and velocities of the links and joints are continuously calculated as the simulation proceeds.

The simulation of the Kin/Geo Robot starts with the initialization of some of its data structures. The initialization involves:

1. the initialization and creation of the link position dictionary, joint position dictionary, link velocity dictionary and joint velocity dictionary.
2. the reading of geometric data of the links and storing into a data structure.
3. reading of kinematic data of the joints and storing into a data structure.

The robot will accept the part when its state is idle and when the former previous object is scheduled to send it. The robot receives the part and schedules the start of the robot program. Each command of the robot program is executed for a certain time interval as specified by it. Before each command is executed the time of occurrence of the next command is scheduled. The instruction pointer is continuously updated.

After the entire program has been executed, the instruction pointer is reset to the beginning of the program and the part is scheduled for departure from the robot.
4.1.3.3 Passive Entity Object

Products are represented as passive entity objects. Each part has a dynamic data structure which allows the part to change its attribute sets depending upon its interaction with active entity objects.

The simulation logic flow adopted by the passive entity objects is different from the active entity objects. The logic flow consists of the manipulation of the attribute sets. Passive entity objects interact with the interaction objects and allow them to: 1. create attribute sets, 2. destroy attribute sets, 3. find values of attributes, and 4. change values of attributes. The logic flow is shown in Figure 19.

The creation of the attribute set essentially involves the development of a database which has a dynamic attribute structure. The database is discussed in detail in Section 4.1.4. This database can be created or destroyed by sending messages to it and executing Smalltalk methods. If attributes need to be changed or altered then the passive entity object identifies the attribute by the part name and then performs the desired operation.

Since a part has a dynamic attribute structure, its attribute set changes depending on the active entity it is with. If the part is with a symbolic entity then, it has symbolic attributes, namely time and state. If it is with a kin/geo object then, it has kin/geo attributes, namely geometry, velocity, and position.
Figure 19. Passive Entity Object Logic Flow
4.1.3.4 Interaction Objects

Interaction objects simulate the interactions between entities within a assembly system. The interaction objects in the simulation are Part Enter, Part Exit, Transfer, Assemble and Collision Check. Each of these interaction objects can be modeled at different levels of abstraction.

The logic flow executed by the interaction objects is similar to the logic flow of the active entity objects. In reality, only the Part Enter interaction object goes through the three phases of the logic flow like the active entity objects. The logic flow is shown in Figure 20. The other interaction objects undergo only the conditional scan phase when the active entity object they are attached to, receives a part. The development of the interaction objects at different levels of abstraction is discussed.

4.1.3.4.1 Enter Object

The Enter object is responsible for the arrival of parts into the system. This object is not responsible for handling the arrival of component parts to be added to the products at the assembly cells. The object specifies the part types entering, their interarrival times, their dimensions, schedule plan and icons. The Enter object set consists of two objects:

1. Discrete Part Enter
2. Kin/Geo Part Enter

The time for the next arrival of each part type is stored in the NextArrival dictionary. The lowest of all times is then removed from the dictionary and stored in the eventChain. The Discrete Part Enter monitors the simTime and the eventChain and checks whether a part arrival has been scheduled at the current simTime. If it has been, then a part instance is created and a data structure assigned to it. This data structure has symbolic attributes of the part, for
example, its name, arrival time, schedule plan and icon type. After the part has been created, the next arrival time of the part type is computed and stored in the nextArrival dictionary.

The Kin/Geo Part Enter object performs in exactly the same manner except that it adds kinematic and geometric attributes also to the part data structure. The attributes added are the location and the dimensions of the part.

### 4.1.3.4.2 Exit Object

The Exit object is responsible for the departure of parts from the system. The Exit object set consists of two objects:

1. Discrete Part Exit
2. Kin/Geo Part Exit

The Discrete Part Exit accepts all parts arriving to it. The database containing the symbolic attributes of the part is removed from it, after the statistics of the object are computed. The Kin/Geo Exit object works in a similar fashion except that the kinematic and geometric attributes of the part are removed. This is done by destroying the instance of the part.

### 4.1.3.4.3 Transfer Object

The Transfer object is responsible for transferring the part between the robot objects and the conveyor objects. The Transfer set contains four objects:

1. Discrete Transfer
2. Geo/Kin Transfer
3. Discrete to Geo/Kin Transfer
4. Geo/Kin to Discrete Transfer
Figure 20. Part Enter Object Logic Flow
Discrete Transfer object handles the transfer of part between a symbolic conveyor and a symbolic robot. The Transfer object checks the eventChain for the part departure time from the robot or conveyor. The arrival of the part to the Transfer object is scheduled at the same time. The data structure of the part arriving is then transferred to the Transfer object. The object compares the schedule plan of the part to the identity of its next objects. If the identity of the next object is the first item on the schedule plan, then the part is scheduled to arrive at that next object at the same time. The part data structure is then transferred to the next object. Geo/Kin Transfer handles transfer of part between a kin/geo conveyor and a kin/geo robot.

Discrete to Kin/Geo Transfer or Kin/Geo to Discrete Transfer objects handle the transfer of parts between a symbolic active entity and a kin/geo active entity object. The attribute set of the part is expanded or contracted respectively. For example, if a Transfer object is attached to a symbolic robot, symbolic conveyor and a geometric robot as shown in Figure 37, then the Transfer object may act as a Discrete Transfer object or a Discrete to Kin/Geo Transfer object, at any given time.

4.1.3.4.4 Assemble Object

The Assemble object is attached to the robot object and is responsible for simulating the assembly of parts. Without the presence of the Assemble object, the robot will not read the robot program and the part will be scheduled to leave at the same time that it entered. There are two objects in the Assemble object set:

1. Discrete Assemble object
2. Kin/Geo Assemble object

The Discrete Assemble object is dependent upon the execution of the symbolic robot. The Assemble object reads the robot program and sets the execution time of the robot as the time
taken to execute the entire robot program. After the execution of the program, the Assemble
object checks the instruction pointer of the program to determine if it has completed the as-
semble operation. If it has, then a symbolic part type is added to the partPositionDict dic-
tionary of the existing part. This process involves the execution of time-based activities and
conditional activities.

The Kin/Geo Assemble object is dependent upon the execution of the kin/geo robot. The
Kin/Geo Assemble object works in a similar manner as the Discrete Assemble object with the
exception that each command of the program is executed and the kinematic and geometric
attributes of the part and robot are updated.

4.1.3.4.5 Collision Check Object

The Collision Detection Check involves the execution of conditional activities. The Collision
Check object can only be used in conjunction with a kin/geo active entity object. If the Colli-
sion Check object is attached to a kin/geo robot then, a collision detection check is done be-
tween the robot gripper and the part.

4.1.4 Data Structures

Various data structures were defined for the storage and manipulation of data for use in the
simulation. The entities and interaction objects are functionally independent of each other,
but share a large amount of data between them through data structures. These data struc-
tures are dynamic in nature i.e., the size of the data structures keeps increasing or decreasing
as the simulation proceeds. Some of the data structures are discussed next with regard to
their function in a simulation.
4.1.4.1 PartList Database

The PartList database is a Smalltalk dictionary which uses a key/value pair for the storage of data. It stores the properties of the part types entering at the part name position for efficient lookup. The PartList database is created when an instance of Enter interaction object is created. Values are filled into the database when the part types are specified using the Enter object menu and submenus.

The properties of the part types stored in the database are its name, dimensions, arrival probability, schedule plan, controller, position and icon form. An example of the database is shown in Table 7. Each of these properties can be retrieved from the database by specifying the name of the part. Each property of the part is further an instance of a different class and Smalltalk provides the flexibility to store them into the database. The method used to store values in the database is PartList at: partName put: (PartType newPartType: partName icons: (FormList at: iconName) runBy: itsSimulator dimensions: dimensionsArray prob: p schList: plan partPosition: partPositionDict). In case any of these values are not available then they are stored at nil. Each of these properties are discussed next.

Dimensions: The dimensions of the part entering the system are specified with the help of Prompters in terms of their length, width and height. Taking these dimensions as the reference, the coordinates of the part are calculated. Further, the edge vectors and normals to the faces are calculated and stored on the hard disk. The corners, edge vectors and face normals are used for geometric collision checks with the conveyor or robot gripper.

Probability: The probability distribution is needed to determine the next arrival time of the part into the system. It is used only when the part has to enter the system and not used anywhere else in the simulation.
## PARTLIST DATABASE

<table>
<thead>
<tr>
<th>NAME</th>
<th>CONT.</th>
<th>ICON</th>
<th>DIMEN.</th>
<th>SCH PLAN</th>
<th>PART POS</th>
<th>ARR PROB</th>
</tr>
</thead>
<tbody>
<tr>
<td>PartA</td>
<td>Simul</td>
<td>IconA</td>
<td>(1.1.1)</td>
<td>file1</td>
<td>(0.0.0)</td>
<td>Const. 5</td>
</tr>
<tr>
<td>PartB</td>
<td>Simul</td>
<td>IconB</td>
<td>(.7.1.1)</td>
<td>file2</td>
<td>(1.2.3)</td>
<td>Exp. 2.5</td>
</tr>
<tr>
<td>PartC</td>
<td>Simul</td>
<td>IconC</td>
<td>(1.2.1.8)</td>
<td>file3</td>
<td>(2.3.6)</td>
<td>Norm. 7</td>
</tr>
</tbody>
</table>

Table 7. PartList Database.
**Schedule Plan:** The schedule plan dictates the sequence of assembly operations required to assemble a product. The schedule plan is stored as an ordered collection. The schedule plan is retrieved from the database by a Transfer interaction object. The first element in the schedule plan is taken out by the method `removeFirst` and compared to the next objects of the Transfer object. If there is a match then the part is allowed to proceed, otherwise the element is returned to the first position by the method `addFirst`. Each part entering the system carries a own schedule plan which changes as the part proceeds through the simulation. If a part does not have a schedule plan, then it is thrown out of the system.

**Icon Form:** The icon form for each part type is retrieved from the FormList database and stored in the PartList database. A user can select the icon accept the default one. When each part enters the system, the PartList database is checked for its icon type.

### 4.1.4.2 Event Chain

The eventChain is the data structure in which all the events are stored. The eventChain is an instance of the SortedCollection Class. All events are added to the eventChain by the method `eventChain add: (Event newFor: anObject at: aTime)` and sorted according to the specified time Atime. This method adds an instance of anObject to the eventChain to be executed at aTime. The events in the eventChain are ordered in increasing order of occurrence time. The next event is inserted based on its occurrence time and the occurrence time and the occurrence times of elements within the eventChain.

Events are retrieved from the eventChain by the method `removeFirst`. This removes the first event of the eventChain. According to the sorting procedure adopted by the eventChain, the time of this event is closest to the current simulated time. The event of the object specified by anObject is executed.
Events can be removed or erased from the eventChain by the method `deschedule: anObject`

This will erase the events of the object specified by anObject. Each object in the eventChain is compared to anObject and when a match is found the event is erased.

If the system is reset, then events in the eventChain are reduced by the current simulated time and the current simulated time is set back to zero.

### 4.1.4.3 FormList

The FormList is a data structure which stores all the icons of the part types created. The FormList is an instance of the Smalltalk Set Class. The data structure allows the addition of icons names to the database. No duplicates are allowed and ordering of the elements are arbitrary in the database. When an icon type is retrieved from the FormList database, the icon name is specified which initiates a search through the database.

### 4.1.4.4 RobotProgList

The RobotProgList data structure is identical to the FormList in structure and operation except that it stores the robot program names instead of icon names.

### 4.1.4.5 SubSystemList

SubSystemList is a data structure in which instances of objects are stored. The SubSystemList is an instance of the Set Class. Instances of the objects are added to this database from the User Interface Module. The user is also prompted for the name under which the model is to be stored. The instance can be retrieved from the SubSystemList and be used in any simulation model.
4.1.5 Development of Objects

Before proceeding with the description of the objects, the kinematic and geometric representation of the active entity and passive entity objects are discussed.

4.1.5.1 Kinematics Representation

The kinematic robot, the kin/geo conveyor and the products exhibit motion thus their kinematic attributes were developed. The development of the kinematic robot involved the consideration of the joint positions, link positions, joint velocities and link velocities. The development of the kin/geo conveyor involved the consideration of the velocity of the conveyor and updating product positions and velocities.

All motions within the assembly system being simulated are considered linear. Robot link motion is represented by with a position vector, \( P \), and a velocity vector, \( V \), which represent the position and the velocity of the link with respect to a general coordinate system.

Further, \( J \) represents the linear displacement of the manipulator joints, and \( Jv \) represents the linear velocities of the joints. Therefore, the link position was computed by:

\[
P1 = T \cdot J + O \quad \text{(1)}
\]

\[
P = P1 + Po \quad \text{(2)}
\]

where

\( T = \) joint transformation matrix

\( O = \) link offset vector

\( Po = \) position vector of link 0
The first equation gives the position of the link with respect to link 0 and the second equation gives the position of the link with respect to the general coordinate system.

The link velocity was computed by:

$$ V = T.Jv \quad \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots (3) $$

In case of the conveyor, the velocity of the conveyor was known and the position of the parts on the conveyor were computed by:

$$ p = v.t \quad \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots (4) $$

where

- $v$ = velocity of the conveyor
- $t$ = time increment
- $p$ = position of the part w.r.t. the conveyor

Data files were created which stored the joint transformation matrix for each link. The linear velocities of the joints are read from a robot program. Robot programs are described in a subsequent section. The initial link offset vector is also set and stored in data files. The link offset vector is continuously updated as a simulation proceeds. Thus, link positions and the link velocities are calculated continuously.

### 4.1.5.2 Geometric Representation

The geometric robot, the kin/geo conveyor and products exhibit geometric characteristics. The element components of these objects are represented as cuboids for the sake of simplicity.
The geometric level of abstraction of the entity objects requires the use of solid models and Boolean routines. Since there were no solid modeling classes available in Smalltalk, it was necessary to develop solids model data structures and routines.

A solid model represents a three dimensional objects by defining the topological relationships between its vertices, edges and surfaces. All geometric objects are represented as cuboids. Figure 21 illustrates a cube and the definition of its 8 vertices, 12 edges and 6 faces.

Boolean operations were developed to determine whether two cuboids intersect. These operations were used to detect collisions between robots, parts and conveyors.

Data files were created for all the six links of the Cartesian robot and the conveyor bed and legs. These data files involved the specification of the corner points, edge vectors, and the face normals of each cube. An example of a data file for a cube is shown in Figure 22. Further, an isometric drawing of all the geometric models created is shown in appendix A.

### 4.1.5.3 Object Development Description

The development of each object in the SPG involved the creation of a class, variables and data structures and methods depending upon the function performed by each object. The development of the classes and the data structures are discussed in previous sections. This section discusses the development of objects with respect to their functional requirements. A description on the development of each object at different levels of detail is discussed next.

### 4.1.5.3.1 Symbolic Conveyor

The symbolic conveyor is simulated as a black box to which parts arrive at a certain time and depart after the elapse of a certain time interval. The most important attribute of the symbolic
Figure 22. Cube Data File
conveyor considered was the travel time. The travel time determines the time for which a part is with the conveyor.

Before each part is passed to the conveyor, a confirmation is requested from it for acceptance. The method accept: aPart is executed. If the current capacity of the conveyor is less than ten, then the conveyor is ready for acceptance. The part is then added to the FIFO queue on the conveyor. Further, the departure of the part is scheduled and the time computed is placed on the eventChain. When the time has elapsed, the event is removed from the eventChain and a message is sent to the moveit method to execute.

The moveit method checks whether the next object can accept a part. This is done by sending a message to the accept: aPart method of the next object and requesting a confirmation. If confirmation is granted then the first part is removed from the queue and sent to the next object. If the next object cannot accept the part then the conveyor is stopped and the parts on the conveyor remain stationary until the next object can accept the part. When, the conveyor is stopped it cannot accept any parts.

Depending upon the number of parts present on the conveyor, its icon keeps changing to depict the current capacity of the conveyor. Once the part has left the conveyor, the conveyor icon again changes to depict the current capacity.

### 4.1.5.3.2 Kin/Geo Conveyor

As mentioned earlier, the Kin/Geo Conveyor used continuous simulation. The position of the parts on the conveyor are updated after a fixed interval of time. The positions of the parts are stored in the partPosOnConv dictionary. Like all dictionaries, the partPosOnConv dictionary has a key/value pair. The number of the part arriving serves as the key and the part position as its value.
Before the conveyor can accept a part, it checks to see if a Collision Check interaction object is attached to it. In case it is, then a geometric intersection check is done between the part and the conveyor bed. Only if there is intersection will the part be accepted.

When a part is accepted by a Kin/Geo Conveyor, the former's position is recorded in the partPosOnConv dictionary. The conveyor cannot accept two parts at the same time. In such a case, one part is chosen at random and thrown out of the system. The positions of the parts are scheduled to be updated after a fixed interval of time. In case, another part arrives to the conveyor before the update time is reached, the movement of all the parts on the conveyor are rescheduled. This is done to make sure that all parts on the conveyor move the same distance and are not moved twice during the time interval.

The distance moved by each part during the fixed interval of time is dependent upon the velocity of the conveyor. The velocity can be changed interactively.

After the part has reached the end of the conveyor, the former is removed and sent to the next object. Also, the part number key and its value are removed from the partPosOnConv dictionary.

### 4.1.5.3.3 Symbolic Robot

The development of a symbolic robot was similar to that of a symbolic conveyor. The major differences between the two are the calculation of execution time and capacity. The execution time of a robot is dependent upon its robot program. The time taken to execute the entire program is taken as the execution time of the robot. Also, the capacity of a robot is one therefore, when it is in the working mode, it cannot accept any more parts.

A user provides each robot with an identification number. This number is checked against the schedule plan of the part before accepting it. Only if there is a match between the identifica-
tion number and the first item on the schedule plan will the part be accepted by the robot. Upon accepting a part, the method `changeStateTo: working` changes the status of the robot from idle to working and also changes the icon of the robot by executing the statement `iconSelected whichOne: state`. The departure time is scheduled depending upon the execution time calculated.

At the departure time, the `moveIt` is executed and the part exits the robot. The status of the robot is changed back to idle by the method `changeStateTo: idle`. The next object attached to the robot is requested to accept the part.

### 4.1.5.3.4 Kin/Geo Robot

The development of a Kin/Geo Robot involved the creation and manipulation of its links and joints. A large number of data files were created to store the data of each link in terms of its part coordinates, edge vectors, face normals and surface vectors. Data files were also created for the joint transformation matrices and their offset vectors.

At the start of the simulation, a kin/geo robot reads all the kinematic and geometric data pertaining to it and stores it into the `kinData` and `geoData` data structures respectively. The execution of the robot proceeds with the movement of the links and joints in space.

A Kin/Geo Robot follows the same procedure as the symbolic robot for accepting a part. Once the part is accepted, the robot starts executing the robot program. Each line of the robot program is read and executed and the future occurrence time of the next line or command is stored in the event list.

If the command is a PMOVE and the robot is geometric then, the positions of all the links are updated. The `dynManipFrame` method calculates and updates the positions and velocities of
all the links. The updated positions and velocities of the links are stored in the posDict and velDict dictionaries respectively.

If the robot is kinematic then, the positions of all the joints are updated. Joint positions are updated by taking into consideration the velocities of the joints. The method executed for this procedure is moveJoint: jointPos and: jointVel time: dt. This method calculates the joint positions for the movement of the joints for the incremental time dt. The calculation procedure is discussed in Section 4.1.5.1.

The robot goes through the robot program, one command at a time, and assembles the parts. The position of the parts are also updated. During the robot program, when the robot gripper grips the part, a geometric intersection check is done between the two. The geometric intersection check is done during the conditional scan to check for collision between the part and the gripper. If there is intersection between the two then, the position of the part is continuously updated along with the movement of the gripper. Once the gripper releases the part, another geometric intersection check is done to confirm that there is no intersection between the two.

4.1.5.3.4.1 Robot Control Representation

A simulated robot, controls its action through the execution of a robot program. A program consists of a set of instructions structured into a N X 7 program, where N is the number of rows of the matrix. To simplify the construction of a robot program for the user, only three different types of instructions are used. These instructions are PMOVE, DELAY and WRITEP. Instructions for sensing the presence of a part are built in directly in the Assemble interaction object. The first column of the matrix represents the instruction number. Columns 2 to 6 are used by the commands and column 7 represents the time taken to execute the command.
The PMOVE command specifies the joint movement of a robot over a certain interval of time and is represented by a ‘1’ in the second column of the matrix. Columns 3 to 4 represent the joint velocities of joint 1 to 4 respectively of the robot. The seventh column represents the time taken to execute the PMOVE command.

The DELAY command is indicated by a ‘2’ in the second column. The movements of links are suspended for a particular time interval specified by the value in the seventh column. The DELAY command is used commonly after the gripper has grasped or released a part. When a DELAY command is executed, joint velocities and link velocities are set to zero.

The WRITEP command is indicated by a ‘3’ in the second column of the robot program. This command is used when a part has to be introduced into the system at the robot. This part is then added to the part flowing through the system. Columns 3 to 5 specify the dimensions of the part in terms of the length, width and height respectively.

An example of a robot program is shown in Figure 23.

4.1.5.3.5 Part Object

The Part object can only be attached to the Enter object. Parts will arrive at the system, if and only if, the part object is attached to the Enter Object.

The development of the Part object involved the creation of the PartList dictionary which is a dynamic data structure. During simulation execution, the part object changes the number and types of attribute sets that it possesses in accordance with its interactions with active entity objects. The part may have symbolic attributes with a symbolic entity and kinematic/geometric attributes with a kinematic/geometric entity. When assigning a new attribute value, the part object cross references attributes, by using the PartList database, whenever it possesses multiple attribute sets. During a cross reference, an object retrieves
Figure 23. Robot Program
attribute values which directly corresponds to the newly assigned value. It then assigns these values to the corresponding attributes within other sets. The creation of the PartList dictionary is discussed in detail in Section 4.1.4.1.

4.1.5.3.6 Part Enter Object

The Part Enter object can receive parts only if the Part object is attached to it. The parts enter the system at the Enter icon according to the previously set probability distribution. The arrival probability of each part type is stored in the arrivalProbabilityDict dictionary by the method arrivalProbabilityDict at: partName put: arrivalProbability. The name of the part is the key to the arrival probability in the dictionary. After the Part Enter object has received the part, the arrival time of the next part is calculated and scheduled. The next arrival time is stored in the nextArrivalDict dictionary by the method nextArrivalDict at: partName put: nextArrival. After the arrival of a part, the nextArrivalDict is scanned to determine which part type is entering the system. The part type having the lowest next arrival time is taken out and scheduled to enter into the system.

Each part type carries with it a database which contains the identity of the part, its icon shape, its schedule list, its dimensions and arrival probability. The icon attached to the Enter icon is requested to accept the part. If the part can be accepted, it is passed to that object. This process is depicted animately by a part icon flowing between the two icons along the connecting line. A part will not be accepted by the next object when the following two conditions exist.

1. If the next object is a robot - The robot may be processing a part already. The robot will accept a part only when it is idle.
2. If the next object is a conveyor - The conveyor may be full. In this system the maximum capacity of the conveyor has been set to ten. If the current capacity of the conveyor is ten, then it will not accept a part.
In such a case, the part is thrown out of the system and is depicted graphically by the appearance of a part icon near the Enter icon.

4.1.5.3.7 Part Exit Object

The part exits the system at the exit icon. The Part Exit object is a sink which accepts all parts. The Part Exit object keeps statistics on the number of parts that have passed through the system and their average time in the system. More than one part can exit the system at the same time.

4.1.5.3.8 Transfer Object

The Transfer object can accept only one part at a time. Only after it has passed the part to the next object can a new part be accepted. The Transfer object checks the schedule plan of all the parts that it accepts. The Transfer object stores instances of its next objects into a convDict and robDict data structures. The convDict dictionary is used for storing instances of conveyors and the robDict dictionary is used for storing instances of robots. Also, the object keeps a database of all its next objects. If there is a match between any one of the next objects and the first item on the schedule plan then, the part is passed on to that object. In case, there is no match, then the part exits the system. If a match is found but, the next object cannot accept it because it is busy, then other objects are checked for acceptance. If none are found then the part is held back with the Transfer object. The part will then be sent when the next object is idle.

Any number of robots can be attached to the Transfer object and each one of them will be checked against the schedule plan of the arriving part.
4.1.5.3.9 Assemble Object

As mentioned in the Rules of Inter-Connection, the Assemble object can only be attached to a Robot object. If it is attached to the latter, then the robot will execute the robot program and then send the part out to the next object. In case the Assemble object is not attached to the Robot object, then the latter will send the part out as soon as it receives it without doing any operation on it.

The Assemble object reads the robot program from the hard disk by executing the method `readDataOf: fileName`. The program is read from the hard disk in the form of an array. This array is converted into a matrix with the method `matrixFromArray:arrayName size: aPoint from: aNumber`. This method constructs a matrix from the array of dimensions N X 7, where N is the number of instructions.

4.1.5.3.10 Collision Check Object

As mentioned later in the Rules of interaction, the Collision Check object can only be attached to the kin/geo robot or conveyor objects. If it is attached to any of the latter then, a geometric intersection check is done to find out whether the part and the conveyor or robot gripper have collided. If there is no collision, then the part will not be transported. In case the Collision Check object is not attached, then the geometric intersection check does not take place and the part is transported directly.

To check for collision detection, a class Geometric was created and the method `collisionCheck: firstObject with: secondObject` was developed to start the checking procedure. The collision check requires the data files of the two objects whose intersection is being checked. The point coordinates, edge vectors, face normals, and surface vectors are calculated for both the objects from their data files and stored in matrices. The face normals of one object are checked for orthogonality with the surface vectors of the other object. The second
check involves an intersection check between the edge vectors of one object with the surface vectors of the other. The intersection check determines whether an edge normally intersects a face and also the point of intersection, if any.
4.2 Model Management

A graphical user interface was created to allow a user to construct a model and to control its simulation. It also provides a user with data on an executing or executed model. The development of the user interface is discussed next followed by a description on model construction, simulation control and data analysis.

4.2.1 User Interface Development

The user interface provides a user with an extensive set of tools comprising of panes, icons, menus and their functions, to create a simulation model, save it, and retrieve it from the model base. The development of these tools is discussed next.

4.2.1.1 Creation of Icons

Icons representing the entity and interaction objects were created using the FreeDrawing Class. Smalltalk provides extensive capabilities for screen bit editing. When the expression FreeDrawing new is evaluated, an Icon Editor Pane pops up. This pane allows the creation of a figure using the set of tools provided in the pane menu. The pane menu also provides bit editing capabilities. To bit edit a small portion of the pane a user selects bit edit from the pane menu. The user is then prompted for the size of the screen to be edited. A Bit Editor pane pops up which has a set of two panes. One pane has a palette of sixteen colors and the other has a grid which represents the number of bits being edited. Each square of the grid represents a bit. The icon is created by selecting colors from the color palette by clicking the
left button of the mouse on the color choice. The selection of a square on the grid changes its color to the selected color. To erase a color, a user selects white from the palette and clicks on the portion to be erased. To help visualize the user about the icon being created, there is a small box at the left corner of the pane which depicts the selection of each square on the grid.

The icon created can be saved and then finally copied into the pictureDictionary database for further retrieval. The copying function is provided in the pane menu.

4.2.1.2 Creation of Panes

Smalltalk provides some capabilities to create panes by using the Pane Class and its subclasses. Six different panes were created for usage by the simulator. Each pane has its own set of menus, dispatchers and controllers. The capabilities provided to each pane are different. Each pane has a top pane to which messages are sent which are further relayed to the controlling class. Each pane is discussed next.

4.2.1.2.1 Icon Pane

The Icon Pane is controlled by the Simulation class. At the time of setup, the Icon Pane is initialized. The Icon Pane is set to occupy 1/5th of the screen. Eight icons representing the entity and interaction objects are read from the FreeDrawing pictureDictionary. These icons were created by using the FreeDrawing Pane and its extensive set of drawing tools. Further, the FreeDrawing Pane gives access to the BitEditor Pane which allows the editing and creation of an icon by manipulation of pixels. This procedure is discussed in detail in Section 4.2.1.1. The icons are initialized and offset to zero and then are further offsetted to occupy a certain position on the Icon Pane. Two columns of four icons each are setup. Further, a rec-
tangle is drawn around each icon. The user can click anywhere in the rectangle to select that icon. As soon as the user clicks on a particular icon, another rectangle is drawn around the previous one to show that the icon has been selected. The icons are indexed from one to eight. Upon selection of any icon, its index value is sent as a message to the Simulation Class. Depending upon the index value, an instance of an object is created.

4.2.1.2.2 Simulation Pane

The Simulation Pane is also controlled by the Simulation Class. This is the pane on which the simulation model is built and animated. It is initialized and setup immediately after the setup of the Icon Pane is done. The Simulation Pane occupies 4/5th of the screen. Since the Simulation Pane is an instance of a GraphPane class, it does not allow text to be written on to it. The Simulation Pane supports all the sixteen colors provided by Smalltalk. The Simulation Pane controls the Simulation Menu which is the main menu for executing and building up the model.

4.2.1.2.3 Statistics Pane

The Statistics Pane is controlled by the SimulationObject Class. The Statistics Pane is an instance of TextPane Class and therefore permits only text to be written on to it. The pane displays the statistics of the entity and the interaction objects.

The Statistics Pane has been provided with features to move, change color, change label, resize the pane, zoom the pane or close the pane. The user can view the statistics on the pane, but not make changes to it interactively.

The Statistics Pane is opened when the View Statistics option is selected from the object menu. The pane pops up and is displayed over the existing panes. At this stage, the contents of the instance variable stats is written on to the Statistics Pane. The Statistics Pane now
becomes the active pane and control is passed to it. A user can scroll the pane to view the statistics.

When the close icon is selected, the pane collapses and the control is passed back to the underlying pane.

4.2.1.2.4 Schedule Pane

The Schedule Pane is an instance of a TextPane Class and is controlled by the Enter Class. It is similar in working to the Statistics Pane except it allows the user to enter text on to it. The schedule plan to be followed by each part entering the system is entered on this pane.

The pane can be initialized and setup on the screen by selecting Schedule Plan from the Part Definition Menu. Initially the pane is a blank one and waits for the user input. The pane provides capabilities of copying, cutting or pasting of text. Further, text can be saved or retrieved to or from a file on the hard disk. The user is prompted for a file name when the save or retrieve option is selected from the pane menu. After a file is saved, it is read into the PartList database. This file is stored into the data structure of each part entering the system.

4.2.1.2.5 Robot Program Pane

The Robot Program Pane is identical to the Schedule Plan except that it is controlled by the Robot Class. The pane takes the robot program inputted by the user and saves it into a file on the hard disk. The program is also saved onto a RobotProgList database.

4.2.1.2.6 Icon and Bit Editor Panes

The Icon and Bit Editor Panes are controlled by the Enter Class. Icons for the part types entering into the system are created using these panes. Their working is similar to the working
of the Icon and Bit Editor Panes of the FreeDrawing Class as explained in Section 4.2.1.1. All the icons created are saved on the in the FreeDrawing pictureDictionary database and the FormList database. The FormList database is scanned everytime a part enters the system for the icon form type. The icon type is then stored in the data structure of the part.

4.2.1.3 Creation of Menus

To create the simulation model interactively and control it, an extensive set of menus were created using the Menu and the PromptMenu Classes. The creation of a menu involves the setting of the menu labels and their selectors. Each selector represents a method which is executed when its option is selected. All menus have been created to pop up at the cursor position. A green bar moves over the options and can be selected by clicking the left button of the mouse. Except for the Simulation Menu, all menus pop up due to the result of some ongoing process. The Simulation Menu pops up when the right button of the mouse is clicked in the Simulation Pane.

The different menus created were:

1. Simulation Menu
2. Object Menus
3. Abstraction Level Menu
4. PartList Menu
5. Part Definition Menu
6. FormList Menu
7. Schedule Pane Menu
8. Probability Distribution Menu
9. Robot Program Menu
10. Link Data Menu
11. Conveyor Data Menu
12. Model Base Menu

The function of the Simulation Menu and the Object Menus are discussed next.

### 4.2.1.3.1 Simulation Menu

The Simulation Menu (Figure 24) has commands to execute a simulation, build a model, re-draw the screen, reset the time, view the database and save/retrieve to/from the model base.

To pop up the Simulation Menu, a user moves the cursor into the Graph Pane and clicks on the right button. Any function can be selected by moving the cursor to that function and clicking the left button.

### 4.2.1.3.2 Object Menu

The Object Menu has commands to attach icons together, detach them, or remove them from a simulation model. Depending upon the object, the menu may have more functions that are characteristic of the object. For example, robot program and robot number are additional functions on the robot menu while part type is an additional function on the Enter Menu.

To access the Object Menu, a user pops up the Simulation Menu as explained above. From the Simulation Menu, a user selects the option build up and then click on the icon whose menu needs to be viewed. Each click of the mouse will result in a search through the iconsSelectedTillNow database to find the icon present at the clicked position. If an icon is found, its menu pops up.
Figure 24. Simulation Menu
The Robot Program Menu, the PartList Menu and the FormList Menu are unique in that their menu options keep changing according to the number of elements in their respective databases.

4.2.1.4 Animation

A simulation model is automatically animated when the execution of the model is started. Parts flowing within the system and the changing of the icons depict the action of the system. The conveyor icon changes according to its current capacity to show whether it is empty or 25% full or 50% full or 75% full or 100% full. Also, a robot icon changes when its status changes from idle to working and vice versa. Further, whenever an object receives a part it flashes. All these features have been incorporated into the system to help simulation visualization.

The major class methods used for animation are discussed next.

appearAt: aPosition for: aName: This method displays the part icon specified by aName, at the position aPosition. aPosition is specified as x @ y. This method is used when the part enters the system at the Enter object.

disappearAt: aPosition for: aName: The disappearAt: aPosition for: aName method is similar to the appearAt: aPosition for: aName except in usage. It is used when the next object cannot accept the part and the part has to be thrown out of the system. The part is then displayed at any position near the icon.

quickFrom: firstPoint to: secondPoint for: aName: The quickFrom: firstPoint to: secondPoint for: aName is executed when a part is sent from one object to another. After the receiving
object has approved the acceptance of the part, the latter is shown graphically to pass from
the firstPoint to the secondPoint along the line connecting the two objects. The speed with
which the part travels along this line can also be adjusted.

**flash**: The flashing or blinking of the icon is done when its object receives or sends a part.
The position of the icon is found from the iconsSelectedTillNow database. The icon is then
erased and redrawn to the same position. This simulates the flashing of the icon.

**whichOne**: As the capacity of the conveyor is increased or decreased and the status of the
robot changed, the whichOne method is executed to show a change in icon shape. The pre-
vious icon is erased and the new icon is drawn at that position to reflect the change.

### 4.2.2 Model Construction

#### 4.2.2.1 System Initialization

The System Menu (see Figure 25) is used to open the SPG window. An additional command
**simulate** has been incorporated for doing so.

The SPG window (Figure 26) opens up when the **simulate** option is selected from the System
Menu. The window can be closed or collapsed by selecting the icons on the label bar of the
window. This window is composed of two panes, the Graph Pane and the Icon Pane. The
Graph Pane and the Icon Pane are initialized and set up. Both of them have their own dis-
patchers, menu systems and controllers. The icons are then read from the FreeDrawing
pictureDictionary, initialized and setup in the Icon Pane. The icons, placed in the Icon Pane,
are enclosed in rectangles. The user can select an icon by clicking anywhere within the rec-
tangle. Eight different types of icons are setup in the Icon Pane. These icons represent the active entities, the passive entities and the interaction objects in the assembly system.

During this stage, the data files of the links of the robot and the conveyor are read, translated and stored in the database. After the panes and the icons are initialized, the cursor changes to the cross hair type to show that the system is ready for use. The simulation model can be built by retrieving existing instances of the objects from the model base or by creating new ones.

4.2.2.2 Retrieving an instance

To retrieve an instance of an object or model from the model base, the user selects Retrieve an instance from the Simulation Menu. The user is provided with a list of all the models in the model base. Upon selecting any one of them, an instance of that model is created. The instance variables of each component of the model are reset to its originally saved values. Each object of the model is then placed at the mouse, one by one, and the user can select any position on the Graph Pane for them. After the complete model has been retrieved from the model base, the interconnecting lines between the icons are automatically drawn. Hence, all the objects are retrieved from the model base in their saved form.

4.2.2.3 Creating an instance

To create an instance of an object, the icon representing it can be selected from the Icon Pane. Clicking the left button of the mouse, when the cursor is in the rectangle enclosing the icon, will select the object. All instance variables of that object are initialized and the mouse cursor is replaced by the icon. Each movement of the mouse results in a corresponding movement of the icon. The icon is constantly erased from its previous position. The icon can be placed anywhere in the Graph Pane by clicking the left button of the mouse.
Figure 25. System Menu
Creating an instance of a Part object and Assemble, Exit, Collision Check and Transfer interaction objects involves the selection of their respective icons and placing them in the Graph Pane. The menus of these objects is similar to the one in Figure 27. The creation of the Robot, Conveyor entities and Enter interaction object are discussed.

4.2.2.3.1 Creation of a Robot Instance

After a user selects a robot icon, an Abstraction Level Menu (Figure 28) pops up. This menu has functions to allow for the creation of a robot at a particular level of abstraction. If a geometric or kinematic representation is chosen then, the initialization process takes longer because the data files for the robot link geometry must be read and translated. All the instance variables of the robot are initialized after the selection is made.

The initialization of some variable instances has to be done from the robot menu. The robot number and the robot program can be selected from the robot menu. The user is prompted for the number of the robot. For the robot program, the user is given the option to select an existing robot program or to create a new one. To select an existing program, the user can click on the program name in the menu. The program is read from the hard disk and stored as an instance variable. If a new robot program is to be created then a text pane pops up. The robot program can be entered on this pane. The text pane provides a menu which allows the copying, cutting and pasting of text. After the robot program has been created, it can be saved onto the hard disk by using the save option from the pane menu. The user is prompted for a file name and then it is saved. An existing file can also be retrieved from the hard disk. After the robot program is saved, a user can close the pane by clicking on the close icon of the text bar.

After all the instance variables are created, the finish option can be selected from the robot menu.
Figure 27. Part Menu
Figure 28. Abstraction Level Menu
4.2.2.3.2 Creation of a Conveyor Instance

The creation of the conveyor instance is similar to that of the robot. The conveyor menu also allows the setting of the travel time of the symbolic conveyor and the velocity of the kinematic/geometric conveyor (Figure 29). The travel time is set to a probability distribution of constant, uniform, normal or exponential.

4.2.2.3.3 Creation of a Part Enter instance

The creation of the Enter interaction object is more complex than that of the robot or the conveyor. To determine the types of parts entering the system, Part type option is selected from the Enter menu (Figure 30). The PartList database is scanned and all part types already existing pop up in the form of a menu. An existing part type can be selected or a new one created. To create a new part type, a Part Definition Menu pops up (Figure 31). This menu allows the selection of the dimensions of the part entering, its arrival probability, creation of its schedule plan and creation of its icon. Each of these options are discussed below. Also with the help of Promters some instance variables are initialized (Figure 32).

Select Dimensions: Upon selecting the dimensions options from Part Definition Menu, a prompt window appears on the screen which requests the size of the part in terms of the length, width and height. After the system has received the information from the prompter, a datafile is constructed and saved on the hard disk.

Select Arrival Probability: The arrival probability of the different part types can be chosen from a Probability Menu (Figure 33). This menu has commands that allow the user to select from a probability distribution of constant, exponential, normal and uniform. Each of these probability distribution has a default value which can be changed.
Figure 29. Conveyor Menus
Figure 30. Enter Menu
Figure 31. Part Definition Menu
Figure 32. Promters
Figure 33. Arrival Probability Menu
Creation of Schedule Plan: Each part that enters the system will have a schedule plan which consists of a list of robots to which the part has to go to and also the part to be added at each robot. To create a schedule plan, a text pane pops up when the Schedule Plan option is chosen from the Part Definition Menu (Figure 34). This text pane allows the user to enter text into it and provides editing capabilities. After the schedule plan has been created it can be saved into a file to be stored onto the hard disk.

Selection of Icon: To select an icon for the part type, click on select icon on the Part Definition Menu. All the pre-existing part icons in the FormList database pop up in the form of a menu (Figure 35). The user can select an existing icon or create a new one. If the user wants to create a new icon then a FreeDrawPane pops up. The user can draw a new icon with the extensive set of tools provided in this pane. Capabilities of bit editing are also provided to the user. To bit edit a icon select bit edit from the FreeDrawPane Menu. The user is then provided with an adjustable rectangle which can be clicked anywhere on the pane. Upon clicking the left button of the mouse, a BitEditor window opens which allows the editing of the bits of the screen, the size of the rectangle. The user can choose from an assortment of 16 colors to create the icon. The user can click on any color in the color palette and then start clicking on individual bits. Each bit clicked on will acquire the selected color. Once the icon has been created it can be saved into the FormList database (Figure 36).

Storing of Part Type: The part type specified by using the Part Definition Menu functions can be saved into the PartList database by selecting the menu option Save. If any characteristic of the part type is not selected then it will accept a default value before it is saved.

4.2.2.4 Attaching Icons Together

To connect the icons together, the user needs to access the icon object menu. Upon clicking the right button of the mouse, the simulation main menu pops up. The user can select the
Figure 34. Schedule Plan
Figure 36. Icon and Bit Editor
option to access the object menus. Now, the user has to click the left button of the mouse to select the icon to be manipulated. Wherever the user clicks, that position is recorded and the iconsSelectedTillNow dictionary is searched for icons at that position. If the search is positive i.e. the user has clicked on an icon (say icon A) in the Graph Pane, then its object menu pops up. Selecting the Attach To option will provide the user with an arrow and a prompt Click on object to attach to. Again, the click of the left button of the mouse at any position, will initiate a search through the iconsSelectedTillNow dictionary. If there is an icon (say icon B) present at that location then, a line is drawn connecting the two icons together. A line is drawn between the two icons only if the Rules of Interaction are satisfied. If it is not then an error prompt appears. The line symbolizes that there is interaction between the two icons. An instance of icon B is stored in the nextObjects database of icon A and similarly, an instance of icon A is stored in the previousObjects database of icon B. In such a manner, all the selected icons can be connected together in a chain, where each icon knows the identity of its previous and next icons only.

In the case of Transfer icon, it can be connected to more than one icon. Instances of each of these connected to icons are stored in the nextObjSet database of the Transfer object. This database is then used for matching with the schedule list of the part arriving to the Transfer icon. Figure 37 shows some components of the manufacturing system attached to each other.

**4.2.2.4.1 Rules of Interaction**

These are the rules a user should follow to connect the icons together. The SPG automatically checks these rules when any two icons are attached. If the Rules of Interaction are not followed, then an error message is displayed.

1. No object except for a Transfer object can be attached to more than one object,
2. No object can be attached back to its previous object,
3. Only Part object can be attached to Enter object and vice versa,
4. No object can be attached to Part, Collision Check or Assemble objects,
5. A Collision Check object can only be attached to kin/geo conveyor or kin/geo robot,
6. An Exit object cannot be attached to any other object,
7. An Assemble object can only be attached to Robot object, and
8. Two robots or conveyors cannot be attached to each other directly without an interaction object.

4.2.2.5 Detaching Icons

An icon (icon A) can be detached from its next object (icon B) by selecting Detach From from the formers object menu. The connecting line between the two icons is erased and the instance of icon B is removed from the nextObjects database of icon A. Similarly, the instance of icon A is removed from the previousObjects database of icon B.

In the case of a Transfer object, which may be attached to more than one object, an arrow with the prompt Click on object to detach from appears. Similar to the attaching function, every click of the mouse will now result in a search through the iconsSelectedTillNow dictionary for an icon at that position. Once an icon is found, the connecting line between the two icons is erased. Also, the instance of the object is removed from the nextObjSet of the Transfer object.

4.2.2.6 Removing an Icon

To remove an icon from the simulation model, the user can select Remove from the object menu. Upon confirmation from the user the icon is erased. All connecting lines into and out of that icon are erased. Its previousObjects and nextObjects database is set to nil and all its future events stored in the event list are removed.
4.2.2.7 Saving an instance

To create an instance of an object or model and store it into the model base, the user can select **Saving an instance** from the Simulation Menu. An arrow with a prompt **Click on object to include** appears at the cursor. Also, a button **Definition complete** is drawn below the time display. The user can click on any object icon to include into the model base. Every click of the mouse initiates a search through the iconsSelectedTillNow dictionary for a icon at the clicked position. If there is an icon present at the clicked position then, a box is drawn around that icon, symbolizing that the object is ready to be included into the model base. An instance of that object is created and all its instance variables are initialized to their current values.

The user can continue to add objects, one after the other, until the **Definition complete** button is clicked (Figure 38). After the button is clicked the user can name the model. The object or model is then saved into the model base.

4.2.2.8 Redrawing the screen

In case the user wants to discard the current model and start on a new one then, the redraw option can be selected from the main menu. The Graph Pane is redrawn and all the objects and their interconnections are erased.

4.2.2.9 Exiting the SPG

To exit the SPG the user can click on the close icon on the Top Pane. All the objects are then discarded.
Figure 38. Saving Instances
4.2.3 Executing the Model

Unlike an implementation in C, Smalltalk does not generate executable code to execute a program. The execution of the program is done by the creation of instances of objects and by passing messages to it. The methods of an instance receive the message, execute and return an object. The object returned may be an integer, an array, an ordered collection, a dictionary or an instance of a class. This procedure continues until the entire program has executed.

To understand the execution of a program better, a sample method is discussed.

\texttt{edgeFaceStatus: aVector}

\begin{verbatim}
\texttt{z := 0.}
\texttt{w := 1.}
\texttt{b := Simu\texttt{ationObject}} \texttt{edgeLineFaceIntercept: aVector}
\texttt{(b = 0) ifFalse: [}
\texttt{(u > 0) ifTrue: [}
\texttt{(u < 1) ifTrue: [\texttt{\#3].}}
\texttt{].}
\texttt{((self realCompare: u with: z) = 1) ifTrue: [\texttt{\#1].}
\texttt{((self realCompare: u with: w) = 1) ifTrue: [\texttt{\#2].}
\texttt{].}
\texttt{\#0}
\end{verbatim}

The method \texttt{edgeFaceStatus: aVector} is executed when it receives the variable \texttt{aVector}. The method executes by setting the values of some variables and then sending a message.
edgeLineFaceIntercept: to the SimulationObject class with the variable aVector. This message is also the name of a method in the SimulationObject class which is executed. The result returned is assigned to variable b. The message realCompare: with is also sent to the class containing the edgeFaceStatus: method, with the variables u and z. Depending upon the results received by the edgeFaceStatus: method, it returns an integer value to the method that executed it.

The SPG program is started by executing the command Simulation new open. This command creates a new instance of the Simulation class and then executes the method open. The method open sets up the different panes of the SPG. The entire SPG is controlled by this method. The movement and clicking of the mouse is also monitored by this method.

If the left button of the mouse is clicked, over an icon in the Icon Pane, then the method open sends a message to the method newOb: objectName to create an instance of an object. If the right button of the mouse is clicked in the Simulation Pane, then a message is sent to the method menu to pop up the Simulation Menu. The Simulation menu further sends messages to the other methods to create a model and execute it.

### 4.2.4 Simulation Control

The Simulation Program Generator has been designed to execute a model indefinitely until a user stops it. The execution of the model is started by executing the method simulate. This method continuously checks for user input before retrieving an event from the eventChain. If a user clicks the left button of the mouse, then the method thereWasAClick sends a value of true to the simulate method. The execution of the model is then stopped.
After the simulation has been stopped, a user can add objects to the executing model or view the statistics on the components of the model. The simulation time can also be reset by selecting reset time from the Simulation Menu. Once the reset option is selected by the user, the simulation time is set to zero. The execution time of the events in the eventChain is also reset. The lowest event time in the eventChain is taken as reference and set to zero. The rest of the event times are reduced by an equal amount as the reference. The instance variables of all objects in the model are also reset.

When the simulation is restarted, the execution of the model picks up from where it left off.

4.2.5 Data Analysis

The analysis of the data provided and the data generated shows the heterogeneous nature of the SPG. The statistical data on each object in the model can be viewed by stopping the simulation and popping up the Statistics Pane.

4.2.5.1 Data Provided

As explained in Section 4.1, all the attributes and data structures of the entity objects and interaction objects are initialized or created at the time of startup. All the attributes of the entities are set to their default values. Further, during the creation of the model the user provides data on:

1. the number of different part types entering the system,
2. the dimensions of the parts,
3. the schedule plan of the parts,
4. the icons of the parts.
5. the arrival probability of the parts,
6. the travel time on the conveyor,
7. the velocity of the conveyor,
8. the robot program, and
9. the robot identification number.

The data to be provided by the user has been kept to a minimum to simplify the creation of the model. All the data can be entered interactively by using the user interface.

Further, the database of the Simulation Program Generator has the geometries of the robot links and the conveyor bed and legs stored in it. This database is created as soon as the system is started. The geometric data of these objects is read from the data files created on the hard disk. To display the database the user can select View database from the Simulation Menu. The user is then presented with an option to view the data of the robot or the conveyor (Figure 39). A pane (window) is then created to display the data of the object selected.

4.2.5.2 Data Generated

The attributes of the objects can be viewed at any particular time during simulation execution on the statistics pane (Figure 40). The user can select the View Statistics option from the object menu. The Assemble, Collision Check and Transfer interaction objects do not have the option of viewing its database because, their data is not valuable to the simulation. Each object has its own set of attributes which are constantly updated. The data generated for each object is listed in Tables 8 to 10.
Figure 39. Link Data Viewing
Figure 40. Statistics Panes
Data of Part Enter Object

1. Part types arriving
2. Arrival probability of new parts
3. Number of parts received
4. Number of parts sent

Data of Part Exit Object

1. Number of parts received
2. Average time in the system

Table 8. Data Generated of Enter and Exit Interaction Object
Table 9. Data Generated of Conveyor Object

1. Capacity
2. Travel time
3. No. of parts on conveyor
4. Number of parts sent
5. Velocity of conveyor
6. Position of parts

Symbolic Conveyor
Kin/Geo Conveyor
Table 10. Data Generated of Robot Object

1. Execution time
2. Idle time
3. Working time
4. Status of the robot
5. Joint velocities
6. Joint positions
7. Link velocities
8. Link positions
9. Part positions

- Symbolic Robot
- Kinematic Robot
- Geometric Robot
4.3 Conclusions

This chapter described the development of a Simulation Program Generator which permits the modular specification of both homogeneous and heterogeneous models. The objects in the hypothetical manufacturing system were based on GIBSS objects and were classified as active entity objects, passive entity objects and interaction objects.

The SPG was developed in the object-oriented environment, Smalltalk. A model base was created which stores instances of the objects in the model and allows the user to retrieve from it. Further, a user interface was created which provides a user with an extensive set of model creation and execution tools like panes, icons and menus.

The data manipulation module accepts data from the user and provides the results during simulation execution. The data on each entity object is maintained and updated whenever a change in its attributes occur.

The contributions of this research to manufacturing system simulation has been the prototyping of an object-oriented SPG capable of the following:

1. the specification of homogeneous and heterogeneous simulation models,
2. the manipulation of a model base for the storage and retrieval of objects,
3. the development of an user-friendly interface for model creation and execution, and
4. the development of an automatic data collection module.
5 DEMONSTRATION

The Simulation Program Generator created in this research allows the specification of homogeneous and heterogeneous models. This chapter demonstrates the construction of a heterogeneous model and its experimental run. The construction of this model incorporates all of the heterogeneous modeling features of the SPG. Further, the entities of the model are modeled at all the levels of abstraction possible.

5.1 Hypothetical Assembly System

The SPG was tested by creating and executing an hypothetical assembly system consisting of a material handling cell and three assembly cells. The material handling cell was represented by three conveyors and the assembly cell by three robots. The assembly system allows the entry of parts into the system and have some assembly operations performed on it. The simulation model was created using the extensive set of user interactive tools provided by the SPG. The following sections discuss the description of the hypothetical model and its construction and execution.
5.1.1 Model Description

An assembly system was modeled that had a set of three conveyors and three robots to complete an assembly operation on a part. Four different types of parts were allowed to enter the system, each having its own schedule plan, icon type, dimensions and arrival probability. After the parts had passed through the entire system, four different types of products were created.

A flow chart of the routes followed by the parts through the system is shown in Figure 41. The parts enter the system and travel down conveyor 1. Now, depending upon the schedule plan of the parts, they are routed to either conveyor 2, robot 1 or robot 2. If the part is routed to robot 1, then after an assembly operation, it is routed to robot 2. Robot 2 performs another assembly operation and sends the part to either conveyor 3 or robot 3. Again depending upon the schedule plan of the part, it is routed to either of the two.

If the part is routed to robot 3, then a final assembly operation is performed and the part is sent out of the system. In case the part is routed to conveyor 3, then it travels down the conveyor and exits the system. At any stage, if the next object cannot accept the part, then the latter is thrown out of the system.

The conveyors and the robots were modeled at different levels of abstraction to show the heterogeneous modeling capability of the SPG. The abstraction levels of the entity objects were chosen at random and are shown in Figure 41. Further, interaction objects were connected to the entity objects to simulate the interactions between them. The entire model is shown in Figure 42.
Figure 41. Part Flow Chart

Key:
- S - Symbolic
- K - Kinematic
- G - Geometric
- K/G - Kin/Geo
Figure 42. Hypothetical Assembly Model

KEY:
- S - Symbolic
- K - Kinematic
- G - Geometric
- K/G - Kin/Geo
5.1.2 Model Construction

The model was constructed using the extensive set of panes, icons and menus provided by the user interface module of the SPG. The construction of the model was divided into three stages. A brief description of each stage follows:

5.1.2.1 Selection of Objects

The objects used in the model were either retrieved from the model base or created through the selection of icons. The conveyor and the robot objects were retrieved from the model base while, the part object and the interaction objects were created.

5.1.2.1.1 Object Retrieval

To retrieve objects from the model base, the Retrieve Instance option was selected from the Simulation Menu. The selection of this option popped up the Model Base Menu, which showed the objects in the model base. The conveyor and robot objects, at their different levels of abstraction, were then selected and placed on the Simulation Pane. The data structures of these objects which were stored with them were initialized and reset.

5.1.2.1.2 Creating Objects

Icons representing a part object and the interaction objects were selected from the Icon Pane. This was done by clicking the left button of the mouse within the rectangle surrounding the icon. These objects were then placed on the cursor of the mouse. The icon was moved
around the screen with the help of the mouse. It was finally placed at a certain position with respect to the entity object icons, by clicking the left button of the mouse.

5.1.2.2 Connection of Objects

All icons were connected following the Rules of Interaction. The object menus were accessed through the Simulation Menu by selecting **buildup** from the latter menu and then clicking the mouse on the icon of the object to be connected. Selecting the **attach To** function of the object menu presented a prompt **Click on object to attach to.** Clicking on an object connected the two icons together by a black line. The connection procedure was started from the part icon and ended at the exit icon.

5.1.2.3 Specification of Object Attributes

The specification of the attributes of the objects was a major stage in model construction. The data required from the user during the construction of the model were:

1. the name of parts entering the system,
2. the dimensions of the parts,
3. the schedule plan of the parts,
4. the arrival probability of the parts,
5. the icon types of the parts,
6. the travel time of conveyors,
7. the velocity of conveyors,
8. the identity of each robot, and
9. the robot program executed by each robot.
The types of parts entering the system and their schedule plans, icon types, dimensions, and arrival probability was set using the prompts and the Part Definition Menu. The schedule plans for all four parts were created on the Schedule Plan Pane. The icons types were taken directly from the FormList database.

The travel time and velocities of the conveyors was then set by using the Probability Menu and the Object Menus respectively. Now, each robot was given an identification number which corresponded to the schedule plan of the parts. The robot programs for the three robots were then created using the Robot Program Pane.

The names of the parts were set as PartA, PartB, PartC, and PartD. The input data for each part is shown in Tables 11 - 14. The travel time of conveyor 1 was set to a constant value of 4.5 and the that of conveyor 3 to a constant value of 6.3. The velocity of conveyor 2 was set to 10. These numerical values were chosen at random. Further, the three robots were assigned identification numbers of 1, 2, and 3. The robot programs executed by these robots are shown in Figures 43, 44 and 45. Further, the geometry of the conveyor and the robots were taken from the data files of the SPG. Note that user input was kept to a minimum to simplify model creation.

5.1.3 Execution and Control

The execution of the simulation model was started by selecting execute from the Simulation Menu. The execution started with the appearance of PartA at the Enter icon. The part was graphically animated as it passed from the Enter icon to the Conveyor icon and then further to the Transfer icon. Here it was transferred to Robot 1. In the meantime, the time was updated and more parts entered the system. The PartA continued on after the execution of the
<table>
<thead>
<tr>
<th>PartA Data</th>
<th></th>
</tr>
</thead>
</table>
| **DIMENSIONS** | **length = 1**  
**width = 1**  
**height = 1** |
| **SCHEDULE PLAN** | **Robot 1 add Part1**  
**Robot 2 add Part2** |
| **ARRIVAL PROBABILITY** | **Constant 6.0** |
| **ICON TYPE** | **PartFormA** |
| DIMENSIONS | length = 1 |
| width = 1 |
| height = 1 |
| SCHEDULE PLAN | Robot 2 add Part2 |
| Robot 3 add Part3 |
| ARRIVAL PROBABILITY | Normal 5.5 |
| ICON TYPE | PartFormB |

Table 12. PartB Data.
### PartC

<table>
<thead>
<tr>
<th>DIMENSIONS</th>
<th>length = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>width = 1</td>
</tr>
<tr>
<td></td>
<td>height = 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SCHEDULE PLAN</th>
<th>Robot 1 add Part1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Robot 2 add Part2</td>
</tr>
<tr>
<td></td>
<td>Robot 3 add Part3</td>
</tr>
</tbody>
</table>

| ARRIVAL PROBABILITY | Exponential 4.6 |

| ICON TYPE | PartFormC |
Table 14. PartD Data.

<table>
<thead>
<tr>
<th>DIMENSIONS</th>
<th>length = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>width = 1</td>
</tr>
<tr>
<td></td>
<td>height = 1</td>
</tr>
<tr>
<td>SCHEDULE PLAN</td>
<td>Robot 3 add Part3</td>
</tr>
<tr>
<td></td>
<td>Robot 4 add Part4</td>
</tr>
<tr>
<td>ARRIVAL PROBABILITY</td>
<td>Constant 10.1</td>
</tr>
<tr>
<td>ICON TYPE</td>
<td>PartFormD</td>
</tr>
</tbody>
</table>
1311100
2100200.5
3100010.8
4200000.5
5102000.5
6120001.1
7100201.75
810-2000.5
91000-10.8
10200000.5
11100-202.25
121-20001.1

Figure 43. Robot Program 1
Figure 44. Robot Program 2

13 1 1 1 0 0
2 1 0 0 2 0 1
3 1 0 0 0 2 0 2
4 2 0 0 0 0 0 0 5
5 1 0 2 0 0 0 . 5
6 1 2 0 0 0 1
7 1 0 0 2 0 2 . 2 5
8 1 0 0 0 - 2 0 . 2
9 2 0 0 0 0 0 . 5
1 0 1 0 0 - 2 0 3 . 2 5
1 1 1 - 2 0 0 0 1
1311100
2100201
3100020.2
4200000.5
5102000.5
6100202.25
7100020.2
8200000.5
9100203.25
10102000.5

Figure 45. Robot Program 3
robot program and traveled to the other robots as dictated by its schedule plan. The simulation continued with the update of the clock and the animation of the entities.

The simulation was interrupted a number of times to check the statistics on the entities. The execution was stopped by clicking the left button of the mouse. The execution of each object in the model was frozen. The execution was restarted by selecting *execute* from the Simulation Menu. The execution started from where it left off. The statistics at different times is shown in Appendix B. The attribute set or database of the part object keeps expanding or contracting as the execution proceeds.
6 SUMMARY AND RECOMMENDATIONS

6.1 Summary

Computer simulation modeling is currently the most extensively used systems analysis technique applied to manufacturing system design. However, the maintenance and creation of simulation models is labor intensive and error prone. An object-oriented Simulation Program Generator can overcome these problems.

An object-oriented SPG is an interactive software tool that translates a model described by the logical assemblage of objects into the code of a simulation language. It was found that current SPGs do not support the creation of heterogeneous models.

The objective of this research was to prototype an object-oriented SPG capable of supporting the modular specification and generation of heterogeneous models. In particular, this system was developed to create simulation models of flexible assembly systems.

The SPG developed utilized GIBSS based entity and interaction objects. The objects in the system were classified as active entity objects, passive entity objects and interaction objects.
These objects were further sub-divided according to their levels of abstraction. A model base was created for these objects which allowed a user to save or retrieve object instances. In addition, specialized data structures were created for storing attributes of these objects. The most significant of these data structures was the PartList database which allowed a part object to change its attribute set depending upon its level of interaction with an entity object.

A user interface was designed which provided an extensive set of tools for the creation and execution of simulation models. The user interface also permitted graphical animation. A simulation model is specified through the use of icons, panes, and menus. The attributes of the objects can be changed or viewed during simulation execution. In addition, during simulation execution, simulation statistics of the objects were generated with the change in their attributes or data structures.

The use of this SPG drastically reduces the labor time and costs associated with the design and modeling during the multi-stage design of assembly systems. Each object class created was modular in nature and can be expanded easily without much alteration in code. Additional object classes can be added to represent a machine tool and other element of a flexible manufacturing system, without affecting other classes. Also, the model base allows the storage and retrieval of objects for future use.

### 6.2 Recommendations

The Simulation Program Generator developed allows the creation of heterogeneous models of assembly systems using a small set of entities. As mentioned earlier, the intent of this research was to test the feasibility of an object-oriented SPG capable of specifying heterogeneous models, rather than the development of a commercial grade product.
Future research should include the development of the SPG as a commercial product. At the low levels of abstraction, the SPG should create commercial grade simulation models of the manufacturing system entities.

To create a truly comprehensive package, the SPG should allow the user to create more manufacturing system entities such as machine tools, an AS/RS, and automated guided vehicles. This would greatly enhance the usefulness of the SPG.

In addition, further refinements would include the adoption of different types of robots. Presently, only a fixed size Cartesian robot can be instantiated. Robot models can be developed to integrate stress analysis and force analysis.

Also, capabilities of the user interface can be extended to allow the debugging of the robot program interactively with standard robot language like AML/E.

At an advanced stage of development of the SFG, the user interface should depict 3-D images of the entities. Smalltalk’s 3D Editor can be used for this purpose. Currently, animation features include the passing of the parts and the changing of the icons with change in attributes. With the 3-D model, the changes in link positions and part positions can also be depicted.
BIBLIOGRAPHY


BIBLIOGRAPHY 142


Vita

Vikas Gupta was born in Kairana, India on December 31st, 1965. He received his high school diploma from Mayo College in Ajmer, India, in April 1982. Following that, he completed two years of pre-university education in D.A.V. College, and joined the Punjab Engineering College in Chandigarh. He received his Bachelor's degree in Mechanical Engineering in May 1988.

In the Fall of 1988, he joined the Graduate School of Virginia Polytechnic Institute and State University to pursue a Master of Science degree in Industrial & Systems Engineering.

He is a member of IIE and SME.