SIMPACKJ/S: A WEB-ORIENTED TOOLKIT
FOR DISCRETE EVENT SIMULATION

By
MINHO PARK

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2002
To my parents, my fiancé, Suwon, 
and my sisters, Hyewon, Hyekyung, Hyeryun, Hyejung, and Hyejoo
ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Paul A. Fishwick, for his continuous advice, guidance, and support in this research work and for giving me the opportunity to work on this good topic. I am grateful to Dr. Sumi Helal and Dr. Joachim Hammer for serving as my committee members.

I really thank Taewoo Kim and Jinho Lee for their technical help and fruitful discussions. I will never forget their invaluable help.

Last but most importantly, I am deeply indebted to my parents and my fiancé, Suwon, for their encouragement, love and understanding. My gratitude is to them.
TABLE OF CONTENTS

ACKNOWLEDGMENTS ........................................................................................................ iv

LIST OF FIGURES ............................................................................................................. vii

ABSTRACT .......................................................................................................................... xi

CHAPTERS

1 INTRODUCTION .......................................................................................................... 1
   Introduction to SimPack ............................................................................................... 2
   Introduction to SimPackJ/S ......................................................................................... 4
   Organization of This Thesis ....................................................................................... 6

2 TRANSLATION RULES FOR SIMPACKJ/S-JAVASCRIPT .................................... 7
   Overview ..................................................................................................................... 7
   Translation Rules ....................................................................................................... 7
   Naming Convention .................................................................................................... 7
   Basic Type .................................................................................................................. 8
   Struct Type ................................................................................................................ 8
   Array .......................................................................................................................... 9
   Basic type array ......................................................................................................... 9
   Multidimensional arrays ......................................................................................... 10
   Arrays of struct ......................................................................................................... 11
   Constants .................................................................................................................. 11
   Pointer ....................................................................................................................... 12
   Parameter Passing .................................................................................................... 14
   Basic type and struct ............................................................................................... 14
   Pointer ....................................................................................................................... 16
   Limits of Translation ............................................................................................... 18
   File Handling ........................................................................................................... 18
   Goto Statement ......................................................................................................... 18
   Putting It All Together ............................................................................................. 19

3 TRANSLATION RULES FOR SIMPACKJ/S-JAVA ................................................. 22
   Overview ................................................................................................................... 22
   Translation Rules ..................................................................................................... 22

v
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1. The organization of SimPack</td>
<td>3</td>
</tr>
<tr>
<td>1-2. The organization of SimPackJ/S</td>
<td>4</td>
</tr>
<tr>
<td>1-3. The XML-based rube architecture</td>
<td>6</td>
</tr>
<tr>
<td>2-1. Example of naming convention</td>
<td>8</td>
</tr>
<tr>
<td>2-2. Example of converting basic types of C into the Java types</td>
<td>8</td>
</tr>
<tr>
<td>2-3. Example of functions used as structures</td>
<td>9</td>
</tr>
<tr>
<td>2-4. Example of converting <em>arrays</em> of C into <em>arrays</em> of JavaScript</td>
<td>10</td>
</tr>
<tr>
<td>2-5. Example of converting a multidimensional <em>array</em> of C into JavaScript</td>
<td>10</td>
</tr>
<tr>
<td>2-6. Example of converting an <em>array of a struct</em> in C into JavaScript</td>
<td>11</td>
</tr>
<tr>
<td>2-7. <em>Constant</em> declaration</td>
<td>12</td>
</tr>
<tr>
<td>2-8. The example for declaring a multidimensional array with a <em>constant</em></td>
<td>12</td>
</tr>
<tr>
<td>2-9. Rule 1 for pointer translation methods in JavaScript and Java</td>
<td>13</td>
</tr>
<tr>
<td>2-10. Rule 2 for pointer translation methods in JavaScript and Java</td>
<td>13</td>
</tr>
<tr>
<td>2-11. Example of <em>pointer</em> manipulations</td>
<td>14</td>
</tr>
<tr>
<td>2-12. Rule 1 for parameter passing</td>
<td>15</td>
</tr>
<tr>
<td>2-13. Example of parameter passing for basic type</td>
<td>15</td>
</tr>
<tr>
<td>2-14. Rule 2 for parameter passing</td>
<td>15</td>
</tr>
<tr>
<td>2-15. Example of parameter passing for <em>struct</em></td>
<td>15</td>
</tr>
<tr>
<td>2-16. Example of parameter passing – <em>pointer</em></td>
<td>17</td>
</tr>
<tr>
<td>2-17. Example of handling a <em>goto</em> statement</td>
<td>18</td>
</tr>
</tbody>
</table>
2-18. Sample program (C and corresponding JavaScript) .................................................21
3-1. Example of naming convention ..................................................................................22
3-2. Example of converting basic types of C into the Java types ........................................23
3-3. Example of converting a struct in C into a class in Java ...........................................24
3-4. Example of converting arrays of C into arrays of Java .............................................25
3-5. Example of converting multidimensional arrays of C into Java ..............................25
3-6. Example of converting arrays of a struct of C into Java ...........................................26
3-7. Constant declaration ...................................................................................................27
3-8. Rules for pointer translation methods in JavaScript and Java .................................27
3-9. Example of a pointer declaration ...............................................................................28
3-10. Example of pointer manipulations .........................................................................29
3-11. Rules for parameter passing ....................................................................................30
3-12. Example of parameter passing for basic type .........................................................30
3-13. Example of parameter passing for struct ..............................................................31
3-14. Example of parameter passing for a pointer ..........................................................32
3-15. Example for declaring a pointer variable of a struct type ...................................32
3-16. Sample program (C and corresponding Java) ..........................................................35
4-1. deq_for_3d.js ..............................................................................................................38
4-2. A fragment of the VRML program ............................................................................38
4-3. The initial state of the 3D virtual plotter ...................................................................39
4-4. The 3D virtual plotter .................................................................................................40
4-5. The result of the Lorenz System - 1 ..........................................................................40
4-6. The result of the Lorenz System - 2 ..........................................................................41
4-7. The result of the Lorenz System - 3 ..........................................................................41
4-8. Transitions ................................................................................................................42
4-9. The initial state of AWACS FSA model

4-10. S1

4-11. S2 – Color change

4-12. S3 – Turning on radar

4-13. S4 – Communication_1

4-14. S5 – Communication_2

4-15. fsa_for_3d.js

4-16. A fragment of the VRML program

4-17. The result of the AWACS FSA model

D-1. Block.c program

E-1. JavaScript and corresponding Java programs

F-1. An initial status of DEQ

F-2. Example of the Lorenz System

F-3. A JavaScript code for the Lorenz System
Abstract of Thesis Presented to the Graduate School of the University of Florida in Partial Fulfillment of the Requirements for the Degree of Master of Science

SIMPACKJ/S: A WEB-ORIENTED TOOLKIT FOR DISCRETE EVENT SIMULATION

By

MinHo Park

May 2002

Chair: Dr. Paul A. Fishwick
Department: Computer and Information Science and Engineering

SimPackJ/S is the JavaScript and Java version of SimPack, which means SimPackJ/S is a collection of JavaScript and Java libraries and executable programs for computer simulations. The main purpose of creating SimPackJ/S is that I allow existing SimPack users to expand simulation areas and provide future users with a freeware simulation toolkit to simulate and model a system in web environments. One of the goals for this thesis is to introduce SimPackJ/S. The other goal is to propose translation rules for converting C to JavaScript and Java. Most parts demonstrate the translation rules with examples. In addition, I discuss 3D dynamic system models and overview an approach to 3D dynamic systems using SimPackJ/S. I explain an interface between SimPackJ/S and the 3D language--Virtual Reality Modeling Language (VRML). This thesis shows how to translate C to JavaScript and Java and how to utilize SimPackJ/S to a 3D web environment.
CHAPTER 1
INTRODUCTION

Our lifestyle has changed dramatically with the appearance of the Internet. Due to the expansion of the Internet, the boundaries of politics, economics, education, and culture have disappeared. Only 10 years ago, nobody thought of getting an education, operating a business, or appreciating masterpieces with a small “iron box” in his room. The Internet has made the impossible become possible.

Another new world inside computers was created by Internet technology that is encompassing web technology and 3D technology: Cyber world or Virtual world. The idea of infinite possibility in a virtual reality world has its attractions. It is made possible with *Java and JavaScript* representing Internet technologies, *web-browser* standing for web technology, and *3D language*—Virtual Reality Modeling Language (VRML). Computer simulation combined with the new technologies has also greatly changed, and it has created a new term, *web-based simulation*. Web-based simulation represents the connection between the web and the field of simulation [1]. Web-based simulation is the most important part of computer simulation. The main purpose of converting *SimPack of C and C++* into *SimPack of Java and JavaScript* is to provide existing SimPack users with the new version of SimPack to extend their simulation areas, help future new users who want to simulate a system in a web-based environment, and create an XML-based rube architecture [2] using a Multimodel Exchange Model (MXL) [3] and Dynamic eXchange Language (DXL) [4]. SimPackJ/S is laying a foundation for the XML-based
rube architecture. Laying the foundation requires the translation for SimPack to SimPackJ/S.

**Introduction to SimPack**

SimPack is a collection of C and C++ libraries and executable programs for computer simulation [5]. SimPack has been widely distributed, especially in computer science and mathematics, since it was constructed in 1990 as freeware including a wide variety of modeling types, as shown in Figure 1-1. There are four reasons to create SimPack [5]:

1. To increase the variety of available model types:
   
   SimPack provides the Queuing.c file as a library consisting of useful procedures and functions to design a model.

2. To create template algorithms for many cases:
   
   SimPack includes template algorithms in each model directory to assist students and professionals solve a specific problem.

3. To avoid learning a special language syntax:
   
   All libraries and programs in SimPack are coded with C and C++, popular languages among researchers. Without wasting their time to learn a new language syntax, researchers can begin creating simulations.

4. To develop a freeware simulation package.
Figure 1-1. The organization of SimPack
**Introduction to SimPackJ/S**

SimPackJ/S is the Java and JavaScript version of SimPack. In other words, SimPackJ/S is a collection of Java and JavaScript libraries and executable programs for computer simulation. However, SimPackJ/S Version 1.0 includes some parts of SimPack, which means I convert *typical model types for simulation classes at the University of Florida* and *necessary model types for an XML-based rube architecture* for the first stage, as shown in Figure 1-2. SimPackJ/S includes the following models:

- **Declarative Models**: Finite State Automaton, Markov, and Petri nets
- **Functional Models**: Block, Cpudisk, Logic, Network, Qnet, and Queuing library.
- **Constraint Model**: Differential Equation

![Figure 1-2. The organization of SimPackJ/S](image-url)
There are three reasons for converting SimPack to SimPackJ/S:

1. To adapt SimPack to a web-based environment:

   SimPack cannot attract simulationists who want to simulate a system in a 
   web-based environment. Therefore, we need to enable SimPack to adjust to 
   the new circumstances.

2. To provide students with a simulation toolkit for learning 3D dynamic 
   models:

   SimPackJ/S was provided in both the undergraduate and graduate computer 
   simulation classes (CAP 4800 and CAP 5805) for a final project.

3. To use a new modeling and simulation paradigm:

   Recently, in a simulation research group at the University of Florida, the 
   XML-based modeling architecture using the Multimodel Exchange Language 
   and SimPackJ/S was proposed. Another XML-based modeling architecture 
   using Dynamic eXchange Language is currently being developed. This DXL 
   project is also employing SimPackJ/S. Refer to Figure 1-3.
Figure 1-3. The XML-based rube architecture

**Organization of This Thesis**

At the beginning of this chapter I introduce SimPack and SimPackJ/S and state reasons for creating SimPack and issues for converting SimPack to SimPackJ/S. I discuss the rules for translating C into JavaScript in Chapter 2. Translation rules for converting C into Java are also discussed in Chapter 3. In Chapter 4, I present the examples with SimPackJ/S in VRML. The conclusion and related work are discussed in Chapter 5 and 6, respectively. Finally, I provide documents and a source code to understand SimPackJ/S in the Appendix.
CHAPTER 2
TRANSLATION RULES FOR SIMPACKJ/S-JAVASCRIPT

Overview

Microsoft Internet Explorer (IE) 6.0 as a web browser and JavaScript 1.2 for the translation are utilized. In most cases, C language can be translated into JavaScript directly except for the structurally or conceptually different parts, such as “pointer,” a multidimensional “array,” and “struct.” For example, JavaScript does not provide the “pointer” concept. To manage “pointer” in JavaScript, a “pointer object” is created. In the case of the multidimensional “array” and the “struct,” a “function” structure in JavaScript is utilized. In this chapter, we discuss the translation rules for C to JavaScript and limitations of conversion. Finally, a program with the C and JavaScript version is demonstrated to compare the differences between C and JavaScript. I categorize the translation rules for C to JavaScript into several types, such as “Naming convention,” “Basic,” “Struct,” “Array,” “Pointer,” and “Parameter passing.”

Translation Rules

Naming Convention

Most identifiers in C can be converted as they are into JavaScript. If reserved words of JavaScript were used as identifiers in C, we would attach a “_” symbol in front of the identifiers in JavaScript.
Basic Type

An important difference between JavaScript and C is that a variable in JavaScript accepts all data types, which means JavaScript conveniently and automatically converts values from one type to another, as necessary. In JavaScript, after declaring a variable, I assign all data types to the single variable. In Figure 2-2, I assign three data types: “char,” “int,” and “float,” to one variable named all_variable.

Struct Type

I use a keyword “function” in JavaScript to create a structure like “struct” in C. The “function” provides a structure, as well as an original function, in functional programming, which means the “function” can be utilized as an “object” in object-oriented programming. Therefore, I define a “function” to represent a structure. When I
use the “function,” I can create an “object” of the “function” with a “new” operator. In Figure 2-3, “function TOKEN()” and “function ITEM()” are applied as structures. With a “new” operator, we can create the “object” of each “function.”

<table>
<thead>
<tr>
<th>//C</th>
<th>//JavaScript</th>
</tr>
</thead>
</table>
| //C typedef struct {
float attr[MAX_NUM_ATTR];
} TOKEN; |
| function TOKEN() |
| 
{ |
| this.attr = new |
| Array(MAX_NUM_ATTR); |
| } |
| function ITEM() |
| 
{ |
| this._time=0.0; // float |
| this._event=0; // int |
| this.token = new TOKEN(); |
| this.priority; // int |
| } |
| var item1 = new ITEM(); |
| var item2 = new ITEM(); |

Figure 2-3. Example of functions used as structures

**Array**

**Basic type array**

With an “Array” constructor and a “new” operator, I can create an “array” in JavaScript. I can assign all types to the array, because JavaScript is a loosely typed language. Consider the example of Figure 2-4. Note that arrays of “char” in C are converted to a “string” type of JavaScript with a *string-manipulating method* like string library functions in C.
Multimensional arrays

JavaScript does not support multidimensional arrays directly. However, I create the multidimensional arrays with “arrays of an array.” In Figure 2-5, I create an array with a “new” operator and assign each array element to another array constructor. To approach a particular number within the two multidimensional server_info arrays, we would write server_info[x][y].
Arrays of struct

<table>
<thead>
<tr>
<th>//C</th>
<th>//JavaScript</th>
</tr>
</thead>
</table>
| struct tokenstruct {  
  int event;  
  float time;  
  float first_arg;  
  float second_arg;  
} token_list[MAX_TOKENS+1]; | function tokenstruct()  
/  
  this._event; // int  
  this._time; // float  
  this.first_arg; // float  
  this.second_arg; // float  
}  

function tokenstruct_array(length)  
/  
  this.length = length;  
  for(var i=0; i<length; i++)  
    this[i] = new tokenstruct();  
}  

var token_list = new tokenstruct_array(MAX_TOKENS+1); |

Figure 2-6. Example of converting an array of a struct in C into JavaScript

In figure 2-6, token_list is declared as a variable of “arrays” of “struct” in C. On the other hand, in Java, I declare a variable, named token_list, and then create an array construction object, tokenstruct_array(). Finally, I create a structure construction object, tokenstruct(). All objects are created by a “new” keyword.

Constants

I convert constants, defined by “#define” in C, into elements of an array in JavaScript. The name of the array is “constant.” All constants are defined in the constant array in JavaScript. We access all constants with “constant.constant_name.”
```c
//C
#define MAX_TOKENS 20000
#define MAX_SERVER 1000
#define MAX_FACILITIES 1000
int const_var = (int)MAX_SERVER;
```

```javascript
//JavaScript
var constant = {
    MAX_TOKENS : 20000,
    MAX_SERVER : 1000,
    MAX_FACILITIES : 1000
}

var const_var =
    constant.MAX_SERVER;
```

**Figure 2-7. Constant declaration**

```c
//C
typedef struct {
    LIST queue;
    int status;
    char name[25];
    int total_servers;
    int busy_servers;
    float total_busy_time;
    float start_busy_time;
    int preemptions;
    int server_info[MAX_SERVER][2];
} FACILITY;
```

```javascript
//JavaScript
function FACILITY()
{
    this.queue; // LIST
    this.status; // int
    this.name; // char []
    this.total_servers; // int
    this.busy_servers; // int
    this.total_busy_time; // float
    this.start_busy_time; // float
    this.preemptions; // int

    this.server_info = new Array(constant.MAX_SERVER); // int[][]
    for(var i=0; i<constant.MAX_SERVER; i++)
        this.server_info[i] = new Array(2);
}
```

**Figure 2-8. The example for declaring a multidimensional array with a constant**

**Pointer**

Establishing a “pointer” concept in JavaScript is the most interesting and challenging part, since there is no “pointer” in JavaScript. First, I tried to use “reference” in place of “pointer.” However, the “reference” could not solve multidimensional “pointers” like a
“pointer to pointer” in C. “Pointer” in C is a variable that occupies memory space in order to store an address. Therefore, I create a “pointer object” to overcome this problem. I want to introduce “Rules for pointer translation methods in JavaScript and Java” with examples.

**Rule 1.** The ‘*’ a symbol representing the value of the variable pointed by a pointer variable, can be converted to ‘.ptr’.

<table>
<thead>
<tr>
<th>C code</th>
<th>JavaScript code</th>
</tr>
</thead>
<tbody>
<tr>
<td>*tmp</td>
<td>tmp.ptr</td>
</tr>
<tr>
<td>**temp</td>
<td>temp.ptr.ptr    // temp is pointer to pointer</td>
</tr>
</tbody>
</table>

**Rule 2.** Both a direct(.) and an indirect(->) operator in C language can be converted to a ‘.’ in JavaScript and Java.

<table>
<thead>
<tr>
<th>C code</th>
<th>JavaScript code</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmp.field = 3; =&gt; tmp.field = 3; // tmp is a structure</td>
<td></td>
</tr>
<tr>
<td>temp-&gt;field = 3; =&gt; temp.ptr.field = 3; // temp is pointer to struct</td>
<td></td>
</tr>
</tbody>
</table>

In Figure 2-11, I create a “function Pointer ()” to manipulate the “pointer” of C. In case of a “pointer to pointer” of C, I manage with passing a “pointer” constructor with a “new” operator as a parameter to the “pointer” constructor itself, new Pointer (new Pointer ()).
Parameter Passing

The C language provides “call-by-value” concept as a parameter-passing mechanism.

On the other hand, JavaScript supports two parameter-passing mechanisms, “call-by-value” and “call-by-reference.” In JavaScript, “call-by-value” is utilized as a basic type (= a primitive type) parameter-passing method. “Call-by-reference” is used when a parameter is an “object.”

Basic type and struct

Since C provides only a “call-by-value” mechanism, I have to create a “call-by-value” mechanism when C passes “struct” as a parameter. Even though there are additional lines added in JavaScript, I implement a similar “call-by-value” mechanism.

The following rules are for a parameter passing mechanism with examples.
**Rule 1. Basic Type: we utilize the same approach as the C language.**

![Figure 2-12. Rule 1 for parameter passing](image)

//C
```c
schedule(int event, float inter_time)
{
    ..........
}
```

//JavaScript
```javascript
function schedule(int event,
    float inter_time)
{
    ..........
}
```

Figure 2-13. Example of parameter passing for basic type

Rule 2. Struct: before assigning passed values to a parameter, we create parameter variables. We then copy the values of the passed reference to the created variables with a temporary reference.

![Figure 2-14. Rule 2 for parameter passing](image)

```
//C
typedef struct token {
    int id;
    float value;
} TOKEN;

void test(TOKEN token)
{
    ..........
}
```

```
//JavaScript
function TOKEN()
{
    this.id;
    this.value;
}

function test(tmp_param)
{
    var token = new TOKEN();
    token.id = tmp_param.id;
    token.value = tmp_param.value;
    ..........
}
```

Figure 2-15. Example of parameter passing for struct
When C passes “struct” as a parameter, we need additional operations in JavaScript to implement a “call-by-value” mechanism of C. In Figure 2-15, for the statement `void test(TOKEN token)` in C, I create a temporary variable, named `tmp_param`. I assign a `TOKEN constructor` with a “new” operator to a variable, named `token`. Finally, I incorporate passed values to newly created parameter variables.

### Pointer

The same translation scheme as the parameter-passing method for “struct” is utilized, because parameter-passing mechanisms for “struct” and “pointer” in C use “call-by-value.” Figure 2-16 seems like a complicated C program, including “pointer to pointer” and “pointer parameter-passing.” But if we apply the parameter-passing for the “struct” mechanism and the “pointer” scheme, we can convert to JavaScript.
Figure 2-16. Example of parameter passing – pointer

```c
typedef struct {
    float attr[10];
} TOKEN;

typedef struct item {
    float time;
    int event;
    TOKEN token;
    int priority;
} ITEM;

typedef struct calist *calptr;
struct calist {
    ITEM entry;
    calptr next;
};
calptr *calendar;

void count1(calptr *count_event) {
    (*count_event)->event = 5;
}

void count2(struct calist *count_event) {
    count_event->event += 7;
}

struct calist top;
    calptr = &top;
    calendar = &calptr;

    count1(calendar);
    count2(calptr);
```

```javascript
function TOKEN() {
    this.attr = new Array(10);
}

function ITEM() {
    this._time = 0.0; // float
    this._event = 0; // int
    this.token = new TOKEN();
    this.priority; // int
}

function Pointer(ptr) {
    this.ptr = ptr;
}

var calptr = new Pointer(); // calist *

function calist() {
    this.entry = new ITEM();
    this.next = new Pointer();
}

var calendar = new Pointer();

    function count1(tmp_ptr) {
    function count2(tmp_ptr) {
        var count_event = new Pointer();
        count_event.ptr = tmp_ptr;
        count_event.ptr.event += 7;
    }

    var top = new calist();
    calptr.ptr = top;
    calendar.ptr = calptr;

    count1(calendar);
    count2(calptr);
```
Limits of Translation

File Handling

For security reasons, JavaScript does not allow the reading or writing to files. However, it is possible to read or write to a file using “ActiveX” embedded into JavaScript. Unfortunately, we cannot expect JavaScript to handle files like C. In SimPackJ/S, I hardcode input data in each program.

Goto Statement

JavaScript reserves the “goto” keyword for future use. However, it does not currently have a “goto” functionality. Because of that reason, I convert to JavaScript differently.

//C
while (1) {
  if (2*parent > heap_count)
    goto exit;
  else
    child = 2*parent;
  if (child+1 <= heap_count)
    if (heap[child+1].time < heap[child].time)
      child++;
  if (heap[parent].time < heap[child].time)
    goto exit;
  heap_swap(heap[parent], heap[child]);
}

//JavaScript
var temp_true = 1;
while (temp_true) {
  if (2*parent > heap_count)
    { temp_true = 0;
      break; }
  else
    child = 2*parent;
  if (child+1 <= heap_count)
    if (heap[child+1].time < heap[child].time)
      child++;
  if (heap[parent].time < heap[child].time) {
    temp_true = 0;
    break; }
  heap_swap(heap[parent], heap[child]);
  parent = child;
} // end of while

Figure 2-17. Example of handling a goto statement
Putting It All Together

I summarize all translation rules for C and corresponding JavaScript programs discussed in this chapter.

<table>
<thead>
<tr>
<th>//C (Logic.c)</th>
<th>//JavaScript (Logic.js in Logic.html)</th>
</tr>
</thead>
</table>
| `#include <math.h>`  
`#include "../queuing/queuing.h"`  
`#define NUM_BLOCKS 100`  
`#define NUM_TYPES 6`  
`#define MAX_INPUTS 5`  
`#define MAX_OUTPUTS 5`  
`#define MAX_PARAMS 5`  
// Blocks */  
`#define GEN 0`  
`#define AND 1`  
`#define NAND 2`  
`#define OR 3`  
`#define NOR 4`  
`#define INV 5`  
`float f[4], savevar[4];`  
`last_time, delta_time,`  
`delay[NUM_BLOCKS];`  
`int num_blocks, out_block, block_num, event;`  
`int in[NUM_BLOCKS][MAX_INPUTS],`  
`out[NUM_BLOCKS][MAX_OUTPUTS];`  
`int block_type[NUM_BLOCKS];`  
`float param[NUM_BLOCKS][MAX_PARAMS],`  
`end_time;`  
`float value[NUM_BLOCKS];`  
`char type_string[20];`  
`char types[NUM_TYPES][20] =`  
`{ "gen","and","nand","or","nor","inv"};`  
`int param_num[NUM_TYPES] = {0,0,0,0,0,0};`  
`int input_num[NUM_TYPES] = {0,2,2,2,1};`  
`int output_num[NUM_BLOCKS];`  
`TOKEN block_token;`  
| `<script language="javascript" src="queuing.js"> </script>`  
`<script language="javascript">`  
`var constant = {`  
`NUM_BLOCKS : 10,`  
`NUM_TYPES : 6,`  
`MAX_INPUTS : 5,`  
`MAX_OUTPUTS : 5,`  
`MAX_PARAMS : 5,`  
`GEN : 0,`  
`AND : 1,`  
`NAND : 2,`  
`OR : 3,`  
`NOR : 4,`  
`INV : 5`  
`};`  
`var f = new Array(4);`  
`var savevar = new Array(4);`  
`var last_time, delta_time;`  
`var delay = new Array(constant.NUM_BLOCKS);`  
`var num_blocks, out_block, block_num;`  
`var in1 = new Array(constant.NUM_BLOCKS);`  
`var out = new Array(constant.NUM_BLOCKS);`  
`var param = new Array(constant.NUM_BLOCKS);`  
`// for multidimensional arrays...`  
`for(var k=0; k<constant.NUM_BLOCKS; k++) {`  
`  in1[k] = new Array(constant.MAX_INPUTS);`  
`  out[k] = new Array(constant.MAX_OUTPUTS);`  
`  param[k] = new Array(constant.MAX_PARAMS);`  
`}`  
`var block_type = new Array(constant.NUM_BLOCKS);`  
`var type_string;`  
`var types = new Array("gen","and","nand","or","nor","inv");`  
`var param_num = new Array(0,0,0,0,0,0);`  
`var input_num = new Array(0,2,2,2,2,1);`  
`var output_num = new Array(constant.NUM_BLOCKS);`  
`var end_time;`  
`var value = new Array(constant.NUM_BLOCKS);`  
`var block_token = new TOKEN();`  
`var event = new INT();`  

Figure 2-18. Continued
//C (Logic.c)

main()
{
    int i;
    init_simpack(HEAP | EMOVE_DUPLICATES);
    read_network();
    for (i=0;i<num_blocks;i++) {
        value[i] = 0.0;
        /* for all GEN nodes, start scheduling */
        if (block_type[i] == GEN) {
            block_token.attr[0] = (float) i;
            schedule(GEN,0.0,block_token);
        } /* end if */
    } /* end for */
    last_time = 0.0;
    while (sim_time() < end_time) {
        next_event(&event,&block_token);
        if (sim_time() != last_time) {
            printf("%-3f ",last_time);
            for(i=0;i<num_blocks;i++)
                printf("%1d ",(int) value[i]);
            printf("\n");
            last_time = sim_time();
        }
    } /* end while */
} /* end main */

read_network()
{
    int i,j,block_num;
    scanf("%d %d %f",
          &num_blocks,&out_block,&end_time);
    for (i=0;i<num_blocks;i++) {
        scanf("%d %s",&block_num,type_string);
        /* determine numeric type */
        for (j=0;j<NUM_TYPES;j++) {
            if (strcmp(type_string,types[j]) == 0)
                block_type[i] = j;
        } /* end for */
        scanf("%f",&delay[i]);
        for (j=0;j<input_num[block_type[i]];j++)
            scanf("%d",&in[i][j]);
        scanf("%d",&output_num[i]);
        for (j=0;j<output_num[i];j++)
            scanf("%d",&out[i][j]);
        for (j=0;j<param_num[block_type[i]];j++)
            scanf("%f",&param[i][j]);
    } /* end for */
}

//JavaScript (Logic.js in Logic.html)

function main()
{
    var i;
    init_simpack(HEAP | REMOVE_DUPLICATES);
    read_network();
    for (i=0;i<num_blocks;i++) {
        value[i] = 0.0;
        // for all GEN nodes, start scheduling
        if (block_type[i] == constant.GEN) {
            block_token.attr[0] = i;
            schedule(constant.GEN,0.0,block_token);
        } /* end if */
    } /* end for */
    last_time = 0.0;
    while (sim_time() < end_time) {
        next_event(event,block_token);
        if (sim_time() != last_time) {
            document.writeln(last_time);
            for(i=0;i<num_blocks;i++)
                document.writeln(value[i]);
            document.writeln('<br>');
            last_time = sim_time();
        }
    } /* end while */
} /* end of main */

function read_network()
{
    var i,j,block_num;
    num_blocks = 2; out_block = 2;
    end_time = 18.0;
    block_type[0] = 0;
    delay[0] = 4;
    output_num[0] = 1;
    out[0][0] = 1;
    block_type[1] = 5;
    delay[1] = 0;
    in1[1][0] = 0;
    output_num[1] = 0;
} /*end of read_network()*/

Figure 2-18. Continued
//C (Logic.c)

update_block(event, block_token)
int event;
TOKEN block_token;
{
    int j;

    block_num = (int) block_token.attr[0];
    /* determine function type and apply inputs */
    switch (event) {
        case GEN:  value[block_num] = (int)
            (value[block_num]+1) % 2;
            schedule(GEN, delay[block_num], block_token);
            break;
        case AND: value[block_num] = (int)
            block_token.attr[1] &&
            (int) block_token.attr[2];
            break;
        case NAND:  value[block_num] = (((int)
            block_token.attr[1] &&
            (int) block_token.attr[2]) + 1) % 2;
            break;
        case OR:     value[block_num] = (int)
            block_token.attr[1] ||
            (int) block_token.attr[2];
            break;
        case NOR:  value[block_num] = (((int)
            block_token.attr[1] ||
            (int) block_token.attr[2]) + 1) % 2;
            break;
        case INV:    value[block_num] =
            (int)(block_token.attr[1]+1) % 2;
            break;
    } /* end switch */
    /* update all blocks in the out set of this block */
    for(j=0;j<output_num[block_num];j++) {
        block_token.attr[0] = (float) out[block_num][j];
        /* save inputs for this block */
        block_token.attr[1] = value[in1[out[block_num][j]][0]];
        block_token.attr[2] = value[in1[out[block_num][j]][1]];
        schedule(block_type[out[block_num][j]],
            delay[out[block_num][j]], block_token);
    } /* end for */
} /* end update_block */

//JavaScript (Logic.js in Logic.html)

function update_block(event, block_token) {
    var j;

    block_num = block_token.attr[0];
    // determine function type and apply inputs
    switch (event.value) {
        case constant.GEN: value[block_num] =
            (value[block_num]+1) % 2;
            schedule(constant.GEN,
                delay[block_num], block_token);
            break;
        case constant.AND: value[block_num] =
            block_token.attr[1] &&
            block_token.attr[2];
            break;
        case constant.NAND: value[block_num] = ((
            block_token.attr[1] &&
            block_token.attr[2]) + 1) % 2;
            break;
        case constant.OR:     value[block_num] =
            block_token.attr[1] ||
            block_token.attr[2];
            break;
        case constant.NOR:  value[block_num] = ((
            block_token.attr[1] ||
            block_token.attr[2]) + 1) % 2;
            break;
        case constant.INV:    value[block_num] =
            (block_token.attr[1]+1) % 2;
            break;
    } // end of switch
    /* update all blocks in the out set of this block */
    for(j=0;j<output_num[block_num];j++) {
        block_token.attr[0] = out[block_num][j];
        // save inputs for this block
        block_token.attr[1] = value[in1[out[block_num][j]][0]];
        block_token.attr[2] = value[in1[out[block_num][j]][1]];
        schedule(block_type[out[block_num][j]],
            delay[out[block_num][j]], block_token);
    } /* end for */
} /* end update_block */

main();
</script>

Figure 2-18. Sample program (C and corresponding JavaScript)
Overview

For the translation, the 1.02 version of the Java Development Kit (JDK) for Windows NT/Windows 95 is used. Most of the same schemes used in converting C into JavaScript could apply to translating into Java as well, except for the special language syntactic cases. In Java, for example, we properly substitute the keyword “class” for the keyword “function” used in JavaScript. “Procedures” and “functions” used in C are replaced by “methods” in Java. However, in case of a “pointer,” each “pointer object” per type is created since Java is one of the strongly typed languages. On the other hand, just one “pointer object” in JavaScript is created to substitute for the “pointer” concept in C because JavaScript is a loosely typed language. I categorize translation rules for C to Java into several types, such as “Naming Convention,” “Basic,” “Struct,” “Array,” “Constants,” “Pointer,” and “Parameter Passing.”

Translation Rules

Naming Convention

<table>
<thead>
<tr>
<th>//C</th>
<th>//Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>int event;</td>
<td>INT _EVENT;</td>
</tr>
<tr>
<td>int list;</td>
<td>int _list;</td>
</tr>
</tbody>
</table>

Figure 3-1. Example of naming convention
The same scheme employed in JavaScript applies to translation for Java. If reserved words of Java were used as identifiers in C, we would attach a “_” symbol in front of the identifiers in Java.

**Basic Type**

Java is one of the *strongly typed languages*. Every variable must have a declared type. That is, most basic types such as “int,” “char,” “long,” “float,” and “double” in C can be converted into the Java types directly. However, the “string” type in Java can replace the *arrays* of “char” in C. In Figure 3-1, an *integer* type variable named *status* and a *float* type variable named *total_busy_time* in C are directly converted into the Java type of the same name. However, *arrays* of “char” named *status_name* are converted to the “*string*” type of Java.

<table>
<thead>
<tr>
<th>//C</th>
<th>//Java</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int    status;</code></td>
<td><code>int    status;</code></td>
</tr>
<tr>
<td><code>char   status_name[25];</code></td>
<td><code>String  status_name;</code></td>
</tr>
<tr>
<td><code>float  total_busy_time;</code></td>
<td><code>float  total_busy_time;</code></td>
</tr>
</tbody>
</table>

Figure 3-2. Example of converting basic types of C into the Java types

**Struct Type**

In Java, I replace “*struct*” for “*class*” since “*class*” is a set of data and “*methods.*” Once “*class*” is created with a “*new*” operator, we can use and manipulate elements of “*class*” whenever required. Consider the example of Figure 3-3. In the case of *ITEM item1* in C, I first declare a data type--*ITEM*, and a variable--*item1*. I create a “*class*” named *ITEM* to obtain memory with a ”*new*” operator. Moreover, I can access and
handle any data inside the `ITEM` “class.” Note that the constructor inside the “class” of `ITEM` is used for obtaining memory for a `token` variable.

```
//C
typedef struct {
  float attr[MAX_NUM_ATTR];
} TOKEN;

typedef struct item {
  float time;
  int event;
  TOKEN token;
  int priority;
} ITEM;

ITEM item1;
ITEM item2;
```

```
//Java
class TOKEN {
  float attr[];

  TOKEN()
  {
    attr = new float[Const.MAX_NUM_ATTR];
  }
}

class ITEM {
  float time;
  int event;
  TOKEN token;
  int priority;

  ITEM()
  {
    token = new TOKEN();
  }

  ITEM item1 = new ITEM();
  ITEM item2 = new ITEM();
```

Figure 3-3. Example of converting a `struct` in C into a `class` in Java

Array

Basic type array

I translate an `array` defined in C into an `array “object”` with a “new” operator in Java.

For `arrays of “char”` in C, I convert to the “string” type of Java.
Multidimensional arrays

Java supports multidimensional arrays with a form of “arrays of arrays.” I convert to a multidimensional array, as they are in C, and assign the multidimensional array with a “new” operator by specifying the number of elements for each dimension. However, for “arrays of arrays” of “char,” I translate into arrays of “string” in Java. Refer to Figure 3-5.

Arrays of struct

To translate arrays of “struct” in C, I first create “class” with the same name as the “struct” name in C and then place all variables into the “class,” as they are in C. Outside the “class,” I declare a variable with the same name as the name of arrays of “struct” in C and reserve memory spaces with a “new” operator. Finally, I assign physical memory spaces to each element of the arrays. In the example of Figure 3-6, for the “struct” name, tokenstruct, I create a “class,” named tokenstruct, and then place all elements of the “struct” inside the “class.” I declare a variable of arrays, token_list, and reserve memory.
spaces with a “new” operator. To allocate physical memory spaces to each element of the arrays, we utilize a “for” statement.

<table>
<thead>
<tr>
<th>//C</th>
<th>//Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct tokenstruct {</td>
<td>class tokenstruct {</td>
</tr>
<tr>
<td>int event;</td>
<td>int   event;</td>
</tr>
<tr>
<td>float time;</td>
<td>float  time;</td>
</tr>
<tr>
<td>float first_arg;</td>
<td>float  first_arg;</td>
</tr>
<tr>
<td>float second_arg;</td>
<td>float  second_arg;</td>
</tr>
<tr>
<td>} token_list[MAX_TOKENS+1];</td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>public static tokenstruct[] token_list = new tokenstruct[MAX_TOKENS+1];</td>
</tr>
<tr>
<td></td>
<td>..........</td>
</tr>
<tr>
<td></td>
<td>for(i=0;i&lt;MAX_TOKENS;i++) token_list[i] = new tokenstruct();</td>
</tr>
</tbody>
</table>

Figure 3-6. Example of converting arrays of a struct of C into Java

Constants

I convert constants, defined by “#define” in C, into “class” consisting of static final variables named Const. We access all constants with “Const.constant_name.”
Recall “Rules for pointer translation methods in JavaScript and Java.”

**Rule 1.** The ‘*’, a symbol representing the value of the variable pointed by a pointer variable, can be converted to ‘.ptr’.

**Rule 2.** Both a direct(.) and an indirect(->) operator in C language can be converted to a ‘.’ in JavaScript and Java.

I apply the “rules for pointer translation” to a Java environment. The difference between JavaScript and Java related to the pointer concept is that Java has multiple “pointer objects,” because Java is a strongly typed language. On the other hand, JavaScript has just one “pointer object” named “Pointer.” Whenever a pointer with a certain type is declared in the C program, I create a corresponding “pointer object” with the certain type in Java. In the example of Figure 3-9, there are four pointer types: “int,” “TOKEN,”
“NODE,” and “LIST.” I create corresponding “pointer objects” with the specific types: “IntPtrer,” “TokenPointer,” “NodePointer,” and “ListPointer,” respectively.

In Figure 3-10, two “pointer objects,” IntPtrer and Int2Pointer, are used to handle a pointer variable and a pointer-to-pointer variable in C, respectively. To deal with the pointer to pointer scheme in Java, I first declare a basic “pointer object,” IntPtrer, and another “pointer object,” Int2Pointer, representing a “pointer-to-pointer object.”
### Parameter Passing

Java provides two parameter-passing schemes: “call-by-value” and “call-by-reference.” Otherwise, C has one parameter-passing mechanism, “call-by-reference.” In Java, “call-by-value” is utilized when basic type variables are passed. However, “class” variables are passed using “call-by-reference.” I discuss two parameter-passing methods with examples.

```c
//C
main()
{
    int *count;
    int **counter;
    int sum;
    *count = 1;
    **counter = 2;
    sum = (int)*count + **counter;
}
```

```java
//Java
class IntPointer
{   int ptr;
}
class Int2Pointer
{
    IntPointer ptr;
    Int2Pointer()
    {
        ptr = IntPointer;
    }
}
public class Temp
{
    public static void main(String[] args)
    {
        IntPointer count = new IntPointer();
        Int2Pointer counter = new Int2Pointer();
        int sum;

        count.ptr = 1;
        counter.ptr.ptr = 2;
        sum = count.ptr+counter.ptr.ptr;
    }
}
```

Figure 3-10. Example of pointer manipulations
Basic type and struct

We recall *parameter-passing rules for JavaScript and Java.*

<table>
<thead>
<tr>
<th>Rule 1. Basic Type: we utilize the same approach as the C language.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 2. Struct: before assigning passed values to a parameter, we create parameter variables. Then we copy the values of the passed reference to the created variables with a temporary reference.</td>
</tr>
</tbody>
</table>

Figure 3-11. Rules for parameter passing

In the basic type, I convert into Java according to Rule 1. See Figure 3-12.

```c
//C
schedule(int event, float inter_time)
{
    ...........
}
```  

```java
//Java
public static void schedule(int event,
                           float inter_time)
{
    ...........
}
```

Figure 3-12. Example of parameter passing for basic type

When the C program passes “*struct*” as a parameter, we need additional operations in Java to implement a “*call-by-value*” mechanism of C. Consider the example of Figure 3-13. For the statement `heap_insert(ITEM item)` in C, I create a temporary variable, `tmp_item`, with an ITEM type in Java. Subsequently, I create an “ITEM object,” named `item` with an ITEM type, using a “new” operator: `ITEM item = new ITEM()`. Finally, I copy passed values to the newly created variables using `Item_copy “method”`:

```java
Item_copy(item,tmp_item).
```
Figure 3-13. Example of parameter passing for struct

**Pointer**

The same translation scheme as the *parameter-passing method for “struct”* is utilized since *parameter-passing processes for “struct” and “pointer” in C use “call-by-value.”* In Figure 3-14, we see that `event_ptr` is declared as a `pointer` variable in C. I declare a temporary variable, `event_ptr`, and then create an “IntPointer object” with a “new” operator. Finally, I copy a passed pointer value to a newly created pointer variable.

```c
//C
typedef struct item{
    float time;
    int event;
    int priority;
} ITEM;

heap_insert(ITEM item)
{
    //...
}
```

```java
//Java
class ITEM
{
    double time;
    int event;
    int priority;
}

public static void heap_insert(ITEM tmp_item)
{
    ITEM item = new ITEM();
    Item_copy(item,tmp_item);
    //...
}

public static void Item_copy (ITEM Target, ITEM source)
{
    Target.time = source.time;
    Target.event = source.event;
    Target.priority = source.priority;
}
```
The following is an example for when "struct" and "pointer" are used simultaneously.

//C
define struct {
  int size;
} LIST;

create_list(list_ptr)
LIST *list_ptr;
{
  ........
}

//Java
class ListPointer {
  LIST ptr;
}
class LIST {
  int size;
}

public static void create_list(LIST tmp_list_ptr)
{
  ListPointer list_ptr = new ListPointer();
  list_ptr.ptr = tmp_list_ptr;
  ............
}
Putting It All Together

I summarize all translation rules for C and corresponding Java programs that are discussed in this chapter.

```c
//C (CpuDisk.c)

#include "../queuing/queuing.h"
#define n0 6 /* no. class 0 tasks */
#define n1 3 /* no. class 1 tasks */
#define nt n0+n1 /* total no. of tasks */
#define nd 4 /* no. of disk units */
#define qd 1 /* queued req. return */
#define BEGIN_TOUR 1
#define REQUEST_CPU 2
#define RELEASE_CPU 3
#define REQUEST_DISK 4
#define RELEASE_DISK 5

struct token
{
    int cls; /* task’s class (& priority) */
    int un; /* disk for current IO req. */
    double ts; /* tour start time stamp */
} task[nt+1];

TOKEN a_token;

int disk[nd+1], /* disk facility descriptors */
cpu, /* cpu facility descriptor */
ts=500; /* no. of tours to simulate */

double tc[2]={10.0,5.0}, /* class 0,1 mean cpu times */
    td=30.0, sd=2.5; /* disk time mean, std. dev. */

main()
{
    int icount,i,j,event,n[2]; double t,s[2],rn;
    struct token *p;
}
```

```java
//JavaScript (CpuDisk.java)

import SimpackLib.*;
import java.util.*;

class TokenPointer
{
    _token ptr;
}

class _token
{
    int cls;
    int un;
    double ts;
}

class CpuDisk
{
    public static final int n0 = 6;
    public static final int n1 = 3;
    public static final int nt = n0+n1;
    public static final int nd = 4;
    public static final int qd = 1;
    public static final int Const.BEGIN_TOUR = 1;
    public static final int Const.REQUEST_CPU = 2;
    public static final int Const.RELEASE_CPU = 3;
    public static final int Const.REQUEST_DISK = 4;
    public static final int Const.RELEASE_DISK = 5;
    public static int disk[] = new int[nd+1];
    public static int cpu, nts=500, tt;
    public static double td=30.0, sd=2.5;
    public static double[] tc = {10.0, 5.0};
    "public static TOKEN a_token = new TOKEN();"
    public static _token[] task = new _token[nt+1];

    public static void main(String[] args)
    {
        int icount,i,j,event=0;
        int n = new int[2];
        double t,rn;
        double[] s = new double[2];
        TokenPointer p = new TokenPointer();

        for (i=0;i<nt+1;i++) task[i] = new _token();
    }
}
```

Figure 3-16. Continued
//C (CpuDisk.c)

n[0]=n[1]=0; s[0]=s[1]=0.0;
for (i=1; i<=nt; i++) task[i].cls=(i>n0)? 1:0;
init_simpack(LINKED);
cpu=create_facility("CPU",1);
for (i=1; i<=nd; i++)
disk[i]=create_facility("disk",1);
for (i=1; i<=nt; i++)
a_token.attr[0] = (float) i;
schedule(BEGIN_TOUR,0.0,a_token);
} /* end for */
icount = 0;
while (nts>0)
{
icount++;
next_event(&event,&a_token);
i = (int) a_token.attr[0];
p = &task[i];
switch(event)
{
case BEGIN_TOUR: /* begin tour */
a_token.attr[0] = (double) i;
p.ptr.ts=sim_time();
schedule(REQUEST_CPU,0.0,a_token);
update_arrivals();
break;
case REQUEST_CPU: /* request cpu */
j=p.ptr.cls;
a_token.attr[0] = (double) i;
if (preempt(cpu,a_token,j)
== FREE) {
    rn = expntl(tc[j]);
a_token.attr[0] = (double) i;
schedule(RELEASE_CPU,(double) rn,a_token);
} break;
case RELEASE_CPU: /* release cpu, select disk */
a_token.attr[0] = (double) i;
release(cpu,a_token); p->un=random(1,nd);
schedule(REQUEST_DISK,0.0,a_token);
} break;
case REQUEST_DISK: /* request disk */
a_token.attr[0] = (double) i;
if (request(disk[p->un],a_token,0)
== FREE) {
    rn = erlang(td,sd);
a_token.attr[0] = (double) i;
schedule(RELEASE_DISK,
    (double) rn,a_token);
} break;
}

//JavaScript (CpuDisk.java)

n[0]=n[1]=0; s[0]=s[1]=0.0;
for (i=1; i<=nt; i++) task[i].cls=(i>n0)? 1:0;
Queuing.init_simpack(Const.LINKED);
cpu=Queuing.create_facility("CPU",1);
for (i=1; i<=nd; i++)
disk[i]=Queuing.create_facility("disk",1);
for (i=1; i<=nt; i++)
a_token.attr[0] = (double) i;
Queuing.schedule(Const.BEGIN_TOUR,0.0,a_token);
} /* end for */
icount = 0;
while (nts>0)
{
icount++;
tt=Queuing.next_event(event,a_token); event = tt;
i = (int) a_token.attr[0];
p.ptr = task[i];
switch(event)
{
case Const.BEGIN_TOUR: /* begin tour */
a_token.attr[0] = (double) i;
p.ptr.ts=Queuing.sim_time();
Queuing.schedule(Const.REQUEST_CPU,
0.0,a_token);
Queuing.update_arrivals();
break;
case Const.REQUEST_CPU: /* request cpu */
j=p.ptr.cls;
a_token.attr[0] = (double) i;
if (Queuing.preempt(cpu,a_token,j)
== Const.FREE) {
    rn = Queuing.expntl(tc[j]);
a_token.attr[0] = (double) i;
Queuing.schedule(Const.RELEASE_CPU,
    (double) rn,a_token);
} break;
case Const.RELEASE_CPU: /* release cpu, select disk */
a_token.attr[0] = (double) i;
Queuing.release(cpu,a_token); p.ptr.un=Queuing.random_(1,nd);
Queuing.schedule(Const.REQUEST_DISK,
    0.0,a_token);
} break;
case Const.REQUEST_DISK: /* request disk */
a_token.attr[0] = (double) i;
if (Queuing.request(disk[p.ptr.un],a_token,0)
== Const.FREE) {
    rn = Queuing.erlang(td,sd);
a_token.attr[0] = (double) i;
Queuing.schedule(Const.RELEASE_DISK,
    (double) rn,a_token);
} break;

Figure 3-16. Continued
Figure 3-16. Sample program (C and corresponding Java)

//C (CpuDisk.c)
case RELEASE_DISK: /* release disk, end tour */
    a_token.attr[0] = (double) i;
    release(disk[p.ptr.un],a_token);
    j=p.ptr.cls;
    t=Queuing.sim_time();
    s[j]+=t-p.ptr.ts;
    p.ptr.ts=t;
    n[j]++;
    update_completions();
    a_token.attr[0] = (double) i;
    Queuing.schedule(Const.BEGIN_TOUR,0.0,a_token);
    nts--;
    break;
}

report_stats(); printf("\n");
printf("class 0 tour time = %.2f\n",s[0]/n[0]);
printf("class 1 tour time = %.2f\n",s[1]/n[1]);
} /* end of main

Queuing.c

schedule(event,inter_time,token)
int event;
double inter_time;
TOKEN token;
{
    double event_time;
    ITEM an_item;
    int i,token_id;
    token_id = (int) token.attr[0] % MAX_TOKENS;
    event_time = current_time + inter_time;
    if (!(remove_duplicates) ||
        (token_list[token_id].event != event) ||
        (token_list[token_id].time != event_time) ||
        (token_list[token_id].first_arg != token.attr[1]) ||
        (token_list[token_id].second_arg != token.attr[2])) {
        token_list[token_id].event = event;
        token_list[token_id].time = event_time;
        token_list[token_id].first_arg = token.attr[1];
        token_list[token_id].second_arg = token.attr[2];
        an_item.time = event_time;
        an_item.event = event;
        for (i=0;i<MAX_NUM_ATTR;i++)
            an_item.token.attr[i] = token.attr[i];
        switch (event_list_type) {
            case Const.LINKED:
                insert_list(&event_list,&an_item,Const.TIME_KEY);
                break;
            case Const.HEAP:
                heap_insert(an_item);
                break;
        }
        trace_update();
        an_item.time = event_time;
        an_item.event = event;
        for (i=0;i<MAX_NUM_ATTR;i++)
            an_item.token.attr[i] = token.attr[i];
        switch (event_list_type) {
            case Const.LINKED:
                insert_list(event_list,an_item,Const.TIME_KEY);
                break;
            case Const.HEAP:
                heap_insert(an_item);
                break;
        }
    }
} /* end if */
} /* end schedule */

//JavaScript (CpuDisk.java)
case RELEASE_DISK: /* release disk, end tour */
    a_token.attr[0] = (double) i;
    Queuing.release(disk[p.ptr.un],a_token);
    j=p.ptr.cls;
    t=Queuing.sim_time();
    s[j]+=t-p.ptr.ts;
    p.ptr.ts=t;
    n[j]++;
    Queuing.update_completions();
    a_token.attr[0] = (double) i;
    Queuing.schedule(Const.BEGIN_TOUR,0.0,a_token);
    nts--;
    break;
}

Queuing.report_stats(); System.out.println("\n");
System.out.println("class 0 tour time = " + s[0]/n[0]);
System.out.println("class 1 tour time = " + s[1]/n[1]);
} /* end of main method
} /* end of cpudisk class

Queuing.java

public static void schedule(int event,double inter_time,TOKEN token) {
    double event_time;
    ITEM an_item = new ITEM();
    int i,token_id;
    token_id = (int) token.attr[0] % Const.MAX_TOKENS;
    event_time = current_time + inter_time;
    if (!(remove_duplicates) ||
        (token_list[token_id].event != event) ||
        (token_list[token_id].time != event_time) ||
        (token_list[token_id].first_arg != token.attr[1]) ||
        (token_list[token_id].second_arg != token.attr[2])) {
        token_list[token_id].event = event;
        token_list[token_id].time = event_time;
        token_list[token_id].first_arg = token.attr[1];
        token_list[token_id].second_arg = token.attr[2];
        an_item.time = event_time;
        an_item.event = event;
        for (i=0;i<Const.MAX_NUM_ATTR;i++)
            an_item.token.attr[i] = token.attr[i];
        switch (event_list_type) {
            case Const.LINKED:
                insert_list(event_list,an_item,Const.TIME_KEY);
                break;
            case Const.HEAP:
                heap_insert(an_item);
                break;
        }
    }
} /* end if */
} /* end schedule */
CHAPTER 4
3D DYNAMIC EXAMPLES

Overview

I demonstrate 3D examples created with SimPackJ/S and Virtual Reality Modeling Language. This chapter shows how to apply, modify, and employ SimPackJ/S properly in a 3D web environment. Models presented in this chapter are a declarative model and a constraint model. Declarative models consist of states and events. On the other hand, constraint models are defined in terms of equations even though they can also be represented in graph or other forms. Difference and differential equations are the most applicable to simulation since they explicitly involve time as an independent variable [6]. I choose a Finite State Automaton (FSA) model as an example of declarative models and a Differential EQuation (DEQ) model as an example of declarative models. I utilize Virtual Reality Modeling Language as a 3D language. In addition, I use CORTONA as a VRML web browser for Internet Explorer (IE).

DEQ Model

The Lorenz System as a DEQ model is selected [6]. To simulate the Lorenz System in a 3D web environment, we first create a JavaScript code using a DEQ code generator in SimPackJ/S. We combine the JavaScript code with a script node provided by VRML. Processes for creating the JavaScript code are presented in Appendix F. I proceed from the second step: how to combine the JavaScript code with the script node.

We assume that a JavaScript code, generated by the DEQ code generator, is saved as “deq_for_3d.js.” To connect with VRML, we need to include additional functions in the
“deq_for_3d.js.” In Figure 4-1, two functions are inserted, “perform_equation()” and “return().” The role of the “perform_equation()” is calling a main function to generate data, whenever new data is requested; otherwise it calls the function “return().” In case of “return(),” it passes currently saved values to other script nodes that need those values through “ROUTE.” However, I eliminate three lines from the “deq_for_3d.js” to adjust to the VRML environment. The “While” statement in line 24, for example, is removed because I have to generate data per request. And the “for” statement for initializing global variables in line 22 is also eliminated, since we must pass accumulated values in order to simulate the Lorenz System continuously. A print statement, such as “document.writeln,” is useless in VRML. I properly link the data to 3D objects using “ROUTE.” Consequently, I produce a modified “deq_for_3d.js,” as shown in Figure 4-1. A fragment of the VRML program is presented in Figure 4-2 to show a connection between the modified “deq_for_3d.js” and VRML.

```javascript
function perform_equation(value)
{
new_data = value;
if (new_data == TRUE)
{
data_cnt++;
main();
new_data = FALSE;
}
else
return();
}

var in_ = new Array(4);
var out = new Array(4);
var f = new Array(4);
var savevar = new Array(4);
var time=0,
delta_time,num_equations;
var j=1;
for(var k=1;k<10;k++) out[k] = 0;

function main()
{
//for (i=1;i<=1;i++) {
init_conditions(j);
//while(time < 20) {
if (new_data) {
state();
inTEGRATE();
return();
}
```

Figure 4-1. Continued
I create a 3D virtual plotter to describe the Lorenz System in a 3D web environment. An initial state of the 3D virtual plotter is shown in Figure 4-3. There are four objects in the
3D virtual plotter world: a virtual plotter with two arms and the coordinate system, a start button, a line plot, and two text displayers.

![Figure 4-3. The initial state of the 3D virtual plotter](image)

To start the 3D virtual plotter, we press the start button. Once the 3D plotter is started, x and y values are generated by an equation of the Lorenz System, “deq_for_3d.js.” These generated x and y values are passed to two arms searching for a position fixed by generated x and y values within the coordinate system. At the same time, current x and y values are displayed in text areas and two curved lines are built by current x and y values in the line plot. Refer to a second figure, as shown in Figure 4-4.
Figure 4-4. The 3D virtual plotter

The resulting the Lorenz System world created from "deq_for_3d.js" is shown in Figures 4-5, 4-6, and 4-7.

Figure 4-5. The result of the Lorenz System - 1
Figure 4-6. The result of the Lorenz System - 2

Figure 4-7. The result of the Lorenz System - 3
FSA Model

I choose an Airborne Warning And Control System (AWACS) aircraft as a Finite State Automaton (FSA) model. The FSA model consists of states and transitions. We need to decide the number of state and transitions. In the AWACS model, there are five states and five transitions:

- States
  - S1: Ready state
  - S2: Takeoff state
  - S3: Surveillance state
  - S4: Communication_1 state
  - S5: Communication_2 state

- Transitions

Figure 4-8. Transitions

An initial state of the AWACS model is shown in Figure 4-9. There are three objects: five AWACS aircraft, a stop button, and a statistic button. State changes are represented by pulsating the current state and rotating radar. In addition, I include a different behavior in each state. Each state is represented in Figures 4-10, 4-11, 4-12, 4-13, and 4-14, respectively.
Figure 4-9. The initial state of AWACS FSA model

Figure 4-10. S1

Figure 4-11. S2 – Color change
I employ an FSA template in SimPackJ/S to gather statistics of each state in the AWACS FSA model, assuming that an FSA JavaScript code is “fsa_for_3d.js.” To connect with VRML, we modify the FSA code. First, we include 10 short functions, related to statistics, and replace “function init()” with “function fsa_init().” “Function cnt_time1(),” for instance, is used for accumulating relative time on S1 and “function cnt_state1()” is for accumulating frequencies on S1. The other eight functions are the same method.
as the “Function cnt_time1 ()” and “function cnt_state1 ()” of S1. To set new states and transitions of the AWACS FSA, “functin fsa_int()” is utilized. “Function touched()” is used for starting to calculate statistics. Consequently, I produce a modified JavaScript program, as shown in Figure 4-15. Likewise, I present a fragment of the VRML program, as shown in Figure 4-16.
function cnt_time1(value) { tt1=value;}
function cnt_time2(value) { tt2=value;}
function cnt_time3(value) { tt3=value;}
function cnt_time4(value) { tt4=value;}
function cnt_time5(value) { tt5=value;}

function cnt_state1(value) { ss1=value;}
function cnt_state2(value) { ss2=value;}
function cnt_state3(value) { ss3=value;}
function cnt_state4(value) { ss4=value;}
function cnt_state5(value) { ss5=value;}

function fsa_init()
{
    num_states = 5;
    state_time[0] = tt1;
    state_time[1] = tt2;
    state_time[2] = tt3;
    state_time[3] = tt4;
    state_time[4] = tt5;
    transition[0][0].state = 0;
    transition[0][0].time = 0;
    transition[0][1].state = 1;
    transition[0][1].time = 0;
    transition[1][0].state = 1;
    transition[1][0].time = 0;
    transition[1][1].state = 2;
    transition[1][1].time = 0;
    transition[2][0].state = 2;
    transition[2][0].time = 0;
    transition[2][1].state = 3;
    transition[2][1].time = 0;
    transition[3][0].state = 3;
    transition[3][0].time = 0;
    transition[3][1].state = 4;
    transition[3][1].time = 0;
    transition[4][0].state = 4;
    transition[4][0].time = 0;
    transition[4][1].state = 5;
    transition[4][1].time = 0;
    transition[5][0].state = 2;
    transition[5][0].time = 0;
    transition[5][1].state = 99;
    transition[5][1].time = 0;
    state[0] = 'S1';
    state[1] = 'S2';
    state[2] = 'S3';
    state[3] = 'S4';
    state[4] = 'S5';
    state_freq[0]=ss1;
    state_freq[1]=ss2;
    state_freq[2]=ss3;
    state_freq[3]=ss4;
    state_freq[4]=ss5;
    control_length = 4;
}

var MAX_STATES = 50;
var MAX_LINKS = MAX_STATES*MAX_STATES;
var MAX_STRING_LEN = 20;
var NUM_INPUTS = 2;
var MAX_INPUTS = 20;

var transition = new transition_array_1(MAX_STATES);

var state = new Array(MAX_STATES);
var control = new Array(MAX_STATES);
var state_freq = new Array(MAX_STATES);
var link_freq = new Array(MAX_LINKS);
var
index, current_state, num_states, control_length, total,
i;
var row, col, save_state, tr_index;
var current_time;

var state_time = new Array(MAX_STATES);
var total_state_time = new Array(MAX_STATES);

function touched()
{
    main();
}

function transition_struct()
{
    this.state;
    this.time;
}

function transition_array_1(length)
{
    this.length = length;
    for(var i=0; i<length; i++)
        this[i] = new transition_array_2(MAX_INPUTS);
}

function transition_array_2(length)
{
    this.length = length;
    for(var i=0; i<length; i++)
        this[i] = new transition struct();
}

=> continued in the right column.

Figure 4-15. Continued
function main()
{
    //init();
    fsa_init();
    current_time = 0.0;  
    current_state = 0;
    for (i=0;i<num_states;i++) {
        /* state_freq[i] = 0; */
        total_state_time[i] = 0.0;
    }
    for (i=0;i<MAX_LINKS;i++)
        link_freq[i] = 0;
    index = 0;
    print('FSA Simulator Output');
    print('n');
    while (index <= control_length) {
        print('Time: '+current_time+' State: '+state[current_state]);
        /* gather statistics */
        /*state_freq[current_state] += 1;*/
        current_time += state_time[current_state];
        total_state_time[current_state] += state_time[current_state];
        if(index != control_length) {
            current_time += transition[current_state][control[index]].time;
            save_state = current_state;
            current_state = transition[current_state][control[index]].state;
            tr_index = 
            save_state*num_states+current_state;
            link_freq[tr_index] += 1;
        }
        index++;
    }
}

/* continued in the right column.*/

print('nFSA Model Statistics');
print('State Statistics');
print('----------------');
total = 0.0;
for (i=0;i<num_states;i++)
    total = state_freq[i] + total;
print('State  Name  Freq  Time  Freq %  Time % ');
for (i=0;i<num_states;i++) {
    print('  '+i+'     '+state[i]+'     '+state_freq[i]+'    '+total_state_time[i]+'  '+ (state_freq[i]/total)* 100.0+'    '+total_state_time[i]* 100.0/current_time);
}

print('Link Statistics');
print('----------------');
total = 0;
for (i=0;i<MAX_LINKS;i++)
    total += link_freq[i];
for (i=0;i<MAX_LINKS;i++)
    if(link_freq[i] > 0) {
        row = Math.floor(i/num_states);
        col = i%num_states;
        print(' '+i+'     '+row+' -> '+col+'    '+link_freq[i]+'         '+link_freq[i]* 100.0/total);
    }
} /* end main() */

Figure 4-15. fsa_for_3d.js
To produce a result of the FSA model, we press the stop button and then press the statistic button. A console window appears on a screen, as shown in Figure 4-17.
CHAPTER 5
CONCLUSION

My goal is to convert SimPack of C and C++ version into SimPack of JavaScript and Java version that allows existing SimPack users to extend their simulation areas. I also provide future users with a freeware simulation toolkit to model and simulate a system in a web environment. I have built translation rules for C to JavaScript and Java to accomplish that goal, and I displayed 3D dynamic system models containing SimPackJ/S to exhibit an applicable area. In the translation rules, I cannot treat some types in C, such as enumerated types and unions. However, if we use a function or class concept as a structure, we convert to Java or JavaScript. For educational purposes, SimPackJ/S is a useful simulation toolkit to understand modeling and simulation. SimPackJ/S was employed for a final project in simulation classes (CAP 4800 and CAP 5805) at the University of Florida.

Currently, with the JavaScript translation mechanisms, I am researching another part using MathML that is an XML formation for describing mathematics as a basis for machine-to-machine communication [7], XSLT, and JavaScript. In the case of simple MathML equations, I convert to JavaScript using XSLT. I will enlarge the areas of mathematics equations to differential and difference equations.

The combination of "Java and web" or "JavaScript and web" makes simulation models larger, more complicated, and more dynamic. Although SimPackJ/S cannot cover all those models, libraries and templates provided by SimPackJ/S can help to approach
them. SimPackJ/S has a potential for further improvements. We are continuing to
ameliorate SimPackJ/S and introduce new model templates, as necessary.
CHAPTER 6
RELATED WORK

There are several other web-based discrete simulation packages. Simjava is a process-based discrete event simulation package for Java, similar to Jade's Sim++, with animation facilities [8]. A simjava simulation is a set of entities each running in its own threads. These entities are connected together by ports and can communicate with each other by sending and receiving event objects. A central system class controls all the threads, advances the simulation time, and delivers the events. The progress of the simulation is recorded through trace messages produced by the entities and saved in a file.

Simkit is a small set of Java classes for creating discrete event simulation models [9]. The library is used to either implement stand-alone models or Web page applets. Simkit models can easily be implemented as applets and executed in a Web browser.

JSIM is a Java-based simulation and animation environment [10]. In JSIM, simulation models may be built using either the event package (Event-Scheduling Paradigm) or the process package (Process-Interaction Paradigm). In addition, a graphical designer (jmodel package) allows process models to be rapidly built graphically.

SimPackJ/S is an implementation of SimPack using both JavaScript and Java. We felt it necessary to support both languages, since each has a key role to play in our larger system, rube. Java is faster than JavaScript but not well-supported in VRML plugins. Also, both major web browsers, Netscape and Internet Explorer, have good native support for JavaScript.
SimPackJ/S Version 1.0 is available on

http://www.cise.ufl.edu/~fishwick/simpack/SimPackJS
APPENDIX B
SETTING AN ENVIRONMENT FOR SIMPACKJ/S

1. SimPack-JavaScript

   Internet Explorer (IE) 5.0 or higher

   Netscape 6.2

2. SimPack-Java

   Before using SimPack-Java, JDK must be installed.

   I assume that SimPackJ/S is unzipped on C drive.

   • Adding PATH

     SET PATH=C:\jdk\bin;%PATH%

   • Adding CLASSPATH

     SET CLASSPATH=.;C:\SimPackJS
APPENDIX C
EXPLANATION FOR EACH MODEL PROVIDED BY SIMPACKJ/S

Block Model

This functional model has three functions: a spiral accumulator, a lathe and an inspector which are connected in series. A set number of raw parts are sent through the accumulator, and they wait there until the person operating the lathe is ready to accept them. After the parts have been processed through the lathe, they proceed to an inspector who determines the quality of the new processed part. It takes 2.0 seconds to traverse the accumulator. The person operating the lathe takes 2 minutes per part and cannot handle any more than one part at a time (a queue size of zero). The inspector takes 15 seconds and can handle a queue of 3 processed parts. The lathe operator and inspector times are sampled from a normal distribution.

CPU/DISK Model

CPU/DISK represents the simulation of 1 CPU and 4 disks. There are 9 jobs that move through the system. A single job alternately accesses CPU and a disk many times: this simulates the effect of a real program using these types of resources. Of the 9 jobs, there are 6 low priority jobs and 3 high priority jobs.

LOGIC Model

This is a simple digital logic simulator with nominal gate delays. For available gate types, note the program logic.js - it is easy to add new gates after seeing this template.
MARKOV Model

Consider a simple cycle composed of 5 states. This is a Markov implementation of the dining philosophers. We will simulate a set of dining philosophers (numbered 1 through 5) by letting philosopher 1 and 3 eat simultaneously, and then 2 and 4, and so on. We will let each eating action (the time associated with a state) be 3 minutes. A state, then, is specified as a pair (such as (1,3)). We will specify each arc to be 1 minute. The arc represents a probabilistic change in state. Probabilities are shown in the following 5 state Markov process. We can see, for instance, that philosophers 2 and 4 (the second state) have a greater chance of eating again (probability = 0.4) right after their 3 minute eating activity.

NETWORK Model

This code is used to simulate a bi-directional network of nodes. It uses a purely static routing scheme for messages. Each message starts at a node and has a fixed path of arbitrary length through a set of adjacent nodes. Each node has as many transceivers as it has in-links from other nodes connected to it. Each transceiver is modeled as a resource (facility) that contains a queue if messages are waiting.

PETRI-NET Model

This simulates a time Petri network composed of places and transitions. Transitions can take time while places do not take time. The simulator builds an 'outset' table first to reduce the iterative lookup for transitions that can fire.

QNET Model

QNET is a queuing network simulator that takes as its input the topology (and node information) of a queuing network. There are six kinds of blocks available for modeling
using the queuing network simulator: gen, request, release, join, fork and sink. A
combination request/release connection is used to simulate the behavior of a
queue/facility.
APPENDIX D
BLOCK MODEL SIMULATION IN C

#include "./queuing/queuing.h"
define GENERATE_PARTS 1
define REQUEST_ACC 2
define BLOCK_ACC 3
define RELEASE_ACC 4
define REQUEST_LATHE 5
define BLOCK_LATHE 6
define RELEASE_LATHE 7
define REQUEST_INSPECTOR 8
define RELEASE_INSPECTOR 9
define NUM_PARTS 100

TOKEN a_token;

main()
{
    int i, event, id, accumulator, lathe, inspector, count_parts = NUM_PARTS;
    int acc_waiting, lathe_waiting, acc_waiting_id, lathe_waiting_id;
    float lathe_to_release, blocking_time;
    float cost_part_gen, cost_labor_server1, cost_labor_server2;
    float total_cost;

    init_simpack(LINKED);
    accumulator = create_facility("ACC", 1);
    lathe = create_facility("LATHE", 1);
    inspector = create_facility("INSPECTOR", 1);
    for (i = 1; i <= count_parts; i++) {
        a_token.attr[0] = (float) i;
        schedule(GENERATE_PARTS, 0.0, a_token);
    } /* end for */
    count_parts = NUM_PARTS;
    while (count_parts > 0)
    {
        next_event(&event, &a_token);
        id = (int) a_token.attr[0];
        switch(event)
        {
            Continued in the right column.

            case GENERATE_PARTS:
                schedule(REQUEST_ACC, 0.0, a_token);
                update_arrivals();
                break;
            case REQUEST_ACC:
                if (request(accumulator, a_token, 0) == FREE)
                    schedule(BLOCK_ACC, 2.0, a_token);
                break;
            case BLOCK_ACC:
                acc_waiting = TRUE;
                acc_waiting_id = id;
                if (facility_size(lathe) == 0) {
                    acc_waiting = FALSE;
                    schedule(RELEASE_ACC, 0.0, a_token);
                }
                break;
            case RELEASE_ACC:
                acc_waiting = FALSE;
                release(accumulator, a_token);
                schedule(RELEASE_LATHE, 0.0, a_token);
                break;
            case REQUEST_LATHE:
                lathe_waiting = TRUE;
                lathe_waiting_id = id;
                if (facility_size(inspector) <= 3) {
                    lathe_waiting = FALSE;
                    schedule(RELEASE_LATHE, 0.0, a_token);
                }
                break;
            case BLOCK_LATHE:
                lathe_waiting = TRUE;
                lathe_waiting_id = id;
                if (facility_size(inspector) <= 3) {
                    lathe_waiting = FALSE;
                    schedule(RELEASE_LATHE, 0.0, a_token);
                }
                break;
            case RELEASE_LATHE:
                lathe_waiting = FALSE;
                release(lathe, a_token);
                if (acc_waiting) {
                    a_token.attr[0] = (int) acc_waiting_id;
                    schedule(RELEASE_ACC, 0.0, a_token);
                }
Figure D-1. Block.c program
C Output

```
G:\temp\tc3.0\Work>block0
Costs for Part Processing
Cost for part generation: $40.00
Cost of labor for server 1: $51.07
Cost of labor for server 2: $5.76
Total Cost is $96.83

Total Simulation Time: 11880.407227
Total System Arrivals: 100
Total System Completions: 100

System Wide Statistics
---------------------
System Utilization: 70.4%
Arrival Rate: 0.008417, Throughput: 0.008417
Mean Service Time per Token: 83.607269
Mean # of Tokens in System: 50.947643
Mean Residence Time for each Token: 6052.787598

Facility Statistics
-------------------
F 1 (ACC): Idle: 1.2%, Util: 98.8%, Preemptions: 0
F 2 (LATHE): Idle: 0.2%, Util: 99.8%, Preemptions: 0
F 3 (INSPECTOR): Idle: 87.5%, Util: 12.5%, Preemptions: 0
```
APPENDIX E
BLOCK MODEL SIMULATION IN JAVASCRIPT AND JAVA

Figure E-1. Continued

```html
//JavaScript (Block.js in Block.htm)
<script language="javascript" src="Queuing.js">
</script>
<script language="javascript">
var constant = {
    GENERATE_PARTS : 1,
    REQUEST_ACC : 2,
    BLOCK_ACC : 3,
    RELEASE_ACC : 4,
    REQUEST_LATHE : 5,
    BLOCK_LATHE : 6,
    RELEASE_LATHE : 7,
    REQUEST_INSPECTOR : 8,
    RELEASE_INSPECTOR : 9,
    NUM_PARTS : 100
};
var a_token = new TOKEN();
function main() {
    var i, id, accumulator, lathe, inspector,
    count_parts = constant.NUM_PARTS;
    var event = new INT();
    var acc_waiting, lathe_waiting,
    acc_waiting_id, lathe_waiting_id;
    var lathe_to_release, blocking_time;
    var cost_part_gen, cost_labor_server1,
    cost_labor_server2;
    var total_cost;
    init_simpack(constant.LINKED);
    accumulator = create_facility("ACC", 1);
    lathe = create_facility("LATHE", 1);
    inspector = create_facility("INSPECTOR", 1);
    for (i = 1; i <= count_parts; i++) {
        a_token.attr[0] = i;
        schedule(constant.GENERATE_PARTS,
            0.0, a_token);
    } /* end for */
count_parts = constant.NUM_PARTS;
while (count_parts > 0) {
    next_event(event, a_token);
    id = a_token.attr[0];
    switch(event.value) {
```

```java
//Java (Block.java)
import SimpackLib.*;
import java.util.*;
public class Block {
    public static final int GENERATE_PARTS = 1;
    public static final int REQUEST_ACC = 2;
    public static final int BLOCK_ACC = 3;
    public static final int RELEASE_ACC = 4;
    public static final int REQUEST_LATHE = 5;
    public static final int BLOCK_LATHE = 6;
    public static final int RELEASE_LATHE = 7;
    public static final int REQUEST_INSPECTOR = 8;
    public static final int RELEASE_INSPECTOR = 9;
    public static final int NUM_PARTS = 100;
    public static TOKEN a_token = new TOKEN();
    public static void main(String[] args) {
        int i, event = 0, id, accumulator,
        lathe, inspector, count_parts = NUM_PARTS;
        int acc_waiting = 0, lathe_waiting = 0,
        acc_waiting_id = 0, lathe_waiting_id = 0, tt;
        double lathe_to_release, blocking_time;
        double cost_part_gen, cost_labor_server1,
        cost_labor_server2;
        double total_cost;
        queue.init_simpack(eo.LINKED);
        accumulator = create_facility("ACC", 1);
        lathe = create_facility("LATHE", 1);
        inspector = create_facility("INSPECTOR", 1);
        for (i = 1; i <= count_parts; i++) {
            a_token.attr[0] = (double) i;
            queue.schedule(constant.GENERATE_PARTS,
                0.0, a_token);
        } /* end for */
count_parts = NUM_PARTS;
while (count_parts > 0) {
    tt = queue.next_event(event, a_token);
    id = (int) a_token.attr[0];
    switch(event) {
```
Figure E-1. Continued

//JavaScript
case constant.GENERATE_PARTS:
    schedule(constant.REQUEST_ACC, 0.0,a_token);
    update_arrivals();
    break;

case constant.REQUEST_ACC:
    if (request(accumulator,a_token,0) ==
        constant.FREE)
        schedule(constant.BLOCK_ACC, 2.0,a_token);
    break;

case constant.BLOCK_ACC:
    acc_waiting = constant.TRUE;
    acc_waiting_id = id;
    if (facility_size(lathe) == 0) {
        acc_waiting = constant.FALSE;
        schedule(constant.RELEASE_ACC, 0.0,a_token);
    }
    break;

case constant.RELEASE_ACC:
    acc_waiting = constant.FALSE;
    release(accumulator,a_token);
    schedule(constant.REQUEST_LATHE, 0.0,a_token);
    break;

case constant.REQUEST_LATHE:
    if (request(lathe,a_token,0) ==
        constant.FREE)
        schedule(constant.BLOCK_LATHE,
            normal(120.0,3.0),a_token);
    break;

case constant.BLOCK_LATHE:
    lathe_waiting = constant.TRUE;
    lathe_waiting_id = id;
    if (facility_size(inspector) <= 3) {
        lathe_waiting = constant.FALSE;
        schedule(constant.RELEASE_LATHE, 0.0,a_token);
    }
    break;

case constant.RELEASE_LATHE:
    lathe_waiting = constant.FALSE;
    release(lathe,a_token);
    if (acc_waiting) {
        a_token.attr[0] = acc_waiting_id;
        schedule(constant.RELEASE_ACC, 0.0,a_token);
    }

//Java
case GENERATE_PARTS:
    Queuing.schedule(REQUEST_ACC, 0.0,a_token);
    Queuing.update_arrivals();
    break;

case REQUEST_ACC:
    if (Queuing.request(accumulator,a_token,0) ==
        Const.FREE)
        Queuing.schedule(BLOCK_ACC, 2.0,a_token);
    break;

case BLOCK_ACC:
    acc_waiting = Const.true_; 
    acc_waiting_id = id;
    if (Queuing.facility_size(lathe) == 0) {
        acc_waiting = Const.false_; 
        Queuing.schedule(RELEASE_ACC, 0.0,a_token);
    }
    break;

case RELEASE_ACC:
    acc_waiting = Const.false_; 
    Queuing.release(accumulator,a_token);
    Queuing.schedule(REQUEST_LATHE, 0.0,a_token);
    break;

case REQUEST_LATHE:
    if (Queuing.request(lathe,a_token,0) ==
        Const.FREE)
        Queuing.schedule(BLOCK_LATHE,
            Queuing.normal(120.0,3.0),a_token);
    break;

case BLOCK_LATHE:
    lathe_waiting = Const.true_; 
    lathe_waiting_id = id;
    if (Queuing.facility_size(inspector) <= 3) {
        lathe_waiting = Const.false_; 
        Queuing.schedule(RELEASE_LATHE, 0.0,a_token);
    }
    break;

case RELEASE_LATHE:
    lathe_waiting = Const.false_; 
    Queuing.release(lathe,a_token);
    if (acc_waiting==0) {
        a_token.attr[0] = (int) acc_waiting_id;
        Queuing.schedule(RELEASE_ACC, 0.0,a_token);
    }
Figure E-1. JavaScript and corresponding Java programs
JavaScript Output

Costs for Part Processing
Cost for part generation: 40
Cost of labor for server 1: 51.59913147334217
Cost of labor for server 2: 5.757021211987887
Total Cost is 97.35615268583007

<table>
<thead>
<tr>
<th>SimPack SIMULATION REPORT</th>
</tr>
</thead>
</table>

Total Simulation Time: 1200.1542448022328
Total System Arrivals: 100
Total System Completions: 100

System Wide Statistics
System Utilization: 70.01944511541246
Arrival Rate: 0.008332262326537743 Throughput: 0.008332262326537743
Mean Service Time per Token: 8.403413427395922
Mean # of Tokens in System: 50.52590119345366
Mean Residence Time for each Token: 60.53887479978161

Facility Statistics
F 1 (ACC): Idle: 2.133004627337903%, Util: 97.866953766202%, Preemptions: 0
F 2 (LATH): Idle: 0.14354855841119424%, Util: 99.85645144158881%, Preemptions: 0
F 3 (INSPECTOR): Idle: 87.66511146801349%, Util: 12.334888531866513%, Preemptions: 0
Java Output

Command Prompt

G: \SimPackJS\SimPackJava\Block> java Block
Costs for Part Processing
Cost for part generation: 40.0
Cost of labor for server 1: 0.5802403179850215
Cost of labor for server 2: 5.8879868001863907
Total Cost is 46.44814631984893

<table>
<thead>
<tr>
<th>SimPackJS SIMULATION REPORT</th>
</tr>
</thead>
</table>

Total Simulation Time: 1629.9523259261496
Total System Arrivals: 100
Total System Completions: 100
System Wide Statistics
--------------------------
System Utilization: 66.9469055797956
Arrival Rate: 0.06135148765359094 Throughput: 0.0

Mean Service Time per Token: 10.912926446335263
Mean # of Tokens in System: 53.969677112327005
Mean Residence Time for each Token: 879.6806073872068

Facility Statistics
-------------------
F 1 (ACC) : Idle: 0.0%, Util: 100.0%, Preemptions: 0
F 2 (LATHE) : Idle: 91.73193650723191%, Util: 8.268063492768093%, Preemptions: 0
F 3 (INSPECTOR) : Idle: 7.42734675336399%, Util: 92.5726532466306%, Preemptions: 0
APPENDIX F
HOW TO GET A JAVASCRIPT CODE IN DEQ

I assume that SimPackJ/S is installed on G drive.

1. Double click a “DEQ.html” on
   G:\SimPackJS\SimPackJavaScript\Deq\deq.html.

   A window appears on a screen, as shown in Figure F-1.

![Figure F-1. An initial status of DEQ](image-url)

Figure F-1. An initial status of DEQ
2. Enter data related to the Lorenz System on each field.

Consider Figure F-2.

Figure F-2. Example of the Lorenz System
3. Press a *Submit* button on a screen. We can get a JavaScript code to simulate
the Lorenz System. To use the JavaScript code in VRML, we have to copy
and paste it.

![Image of JavaScript code]

Figure F-3. A JavaScript code for the Lorenz System
LIST OF REFERENCES


BIOGRAPHICAL SKETCH

MinHo Park was born on August 25, 1968, in Jejoo-do, Korea. He received his Bachelor of Science degree in computer engineering from Hong-Ik University, Seoul, Korea, in March 1994. After his graduation, he worked in HP, Korea, LG Information and Communication Corporation, Korea Securities Computer Corporation (KOSCOM), and Good-Morning Securities Company, respectively. In 2002, he received his Master of Science degree in computer and information science and engineering at the University of Florida. His major research areas are modeling for computer simulation and web-based simulation.