UNIVERSIDAD CATOLICA ANDRES BELLO

FACULTAD DE INGENIERIA

ESCUELA DE INGENIERIA INFORMATICA

CATEDRA DE ARQUITECTURA DEL COMPUTADOR

Proyecto

Máquina de Estados Finitos

(FSM)

Sección: 26414

Victoria Paciello

Miguel De Olim

Leonardo Ruíz

Preguntas de autoevaluación

1. **¿Qué es una máquina de estados finitos?**

   Una máquina de estados se puede definir como un modelo de comportamiento dentro de un sistema, dichos comportamientos se entienden como estados limitados. Cada uno de los estados determina el tipo de acción que la máquina lleva a cabo. Está formada por estados, transiciones, entradas y salidas.

2. **¿Qué indica la transición?**

   En teoría de autómatas una transición es una función que determina el cambio de estado dentro de un sistema, teniendo como entrada de dicha función el estado actual del sistema y como salida el estado próximo.

3. **¿Qué es la condición de transición?**

   Son los requisitos de la función de transición para lograr el cambio de un estado actual a un próximo estado.

   Las condiciones asociadas a las transiciones de cambio de estados pueden depender tanto de variables externas (entradas) como internas.

   Como ejemplo de la maquina descrita en este proyecto, tenemos dos condiciones, una de ellas sería el estado de “Búsqueda” el cual requiere que la batería este cargada. Como segundo ejemplo tenemos el estado “Muerto” para el cual se requiere que el batería este vacía.

4. **¿De qué manera se puede diagramar una máquina de estados finitos?**

   La representación utilizada generalmente para una máquina de estados finitos es un diagrama de estados, donde se observan por círculos los estados, con su identificación en el medio y unas flechas que indican la transición. Posee acción de ingreso, que es el nuevo estado que llega y se ejecuta una acción, análogamente posee una acción de salida que es el estado por el cual la máquina termina su ejecución.

   A continuación, se presenta un diagrama del proyecto que posee 8 estados diferentes, los cuales son:

   - Búsqueda (Primera acción después de la acción de entrada)
   - Llevar al microondas
   - Devolver

- Nueva búsqueda
- Recargar
- Ir a la base
- Aleatorio
- Muerte (Última acción antes de la acción de salida).

El primer estado para el dron es “Nueva búsqueda” posee la condición de si hay o no comida en espera, en caso de ser Verdadera la condición, la máquina cambiará de estado a “Búsqueda”, siguiendo por las transiciones, se debe cumplir la condición de llegar al objeto para cambiar de estado a “Llevar al microondas” que en caso de estar en camino al microondas se estará contemplando dos condiciones, la primera es la del estado de completitud de la batería y la segunda es si se ha llegado al microondas o no. Luego se debe esperar a que la condición de que la comida este caliente se cumpla para a cambiar al estado “Devolver”, mientras que la comida sigue calentándose se sigue teniendo en cuenta la condición de la batería para hacer la transición al estado “Muerte”. Siguiendo con las transiciones, se tiene ahora el estado “Devolver”, para cambiar de estado sólo se debe cumplir que el dron llegue al destino, devuelva la comida y transaccionará a “Nueva búsqueda” una vez más.

En el caso en el que el estado actual sea “Búsqueda” y la condición anterior no se cumpla (Haber llegado al destino y devolver la comida) y al mismo tiempo se cumple que la batería sea menor a 400 (Batería < 400), debido a esto la máquina cambiará de estado a “Ir a la base” y, en el momento en el que se cumpla la condición de transición de llegar a la base, la máquina cambiará de estado a “Recargar” luego, cuando su condición de transición se cumpla, que es la de batería recargada, vuelve a donde empezó, el estado de “Búsqueda”.

Si en el estado actual “Ir a base” no se cumple la condición de la base, entonces en algún momento se cumplirá que la batería sea igual a cero (Batería = 0) y será la condición para que la máquina cambie de estado a “Muerte”.

El último caso considera que el estado actual es “Nueva búsqueda” y espera que se cumpla la condición en que no haya comida pendiente, esto cambiará el estado a “Aleatorio”, este estado sólo cambiará si se cumple que la batería sea igual a cero (Batería = 0) con esta condición el estado cambiará a “Muerte”.

5. ¿Cómo se indica el estado inicial de la máquina de estados finitos?

Se señaliza en el diagrama de estados como una flecha que no proviene de ningún estado anterior.

6. **¿Para qué sirve una tabla de estados?**

   Para que se pueda identificar de forma precisa las condiciones de transiciones para de un estado a otro. Y ayudar al momento de codificar en la computadora.

7. **¿Por qué usamos un Timer?**

   Empleamos un “timer” de la misma forma que un “clock” en circuito lógico secuencial, esto nos permite en cada n intervalo de tiempo verificar las condiciones para realizar una transición de estados o mantenerlo.

8. **¿De qué otra forma se puede implementar la máquina de estados finitos?**

   La maquina de estados se puede implementar de diferentes formas, como diagramas de estados, tablas de estados o alguna otra herramienta de representación de estados. Siempre con la intención de representar los diferentes estados, transiciones, entradas y salidas.

9. **¿Cómo puede modificarse el código para que el robot recoja el objeto más cercano en lugar del último de la fila?**

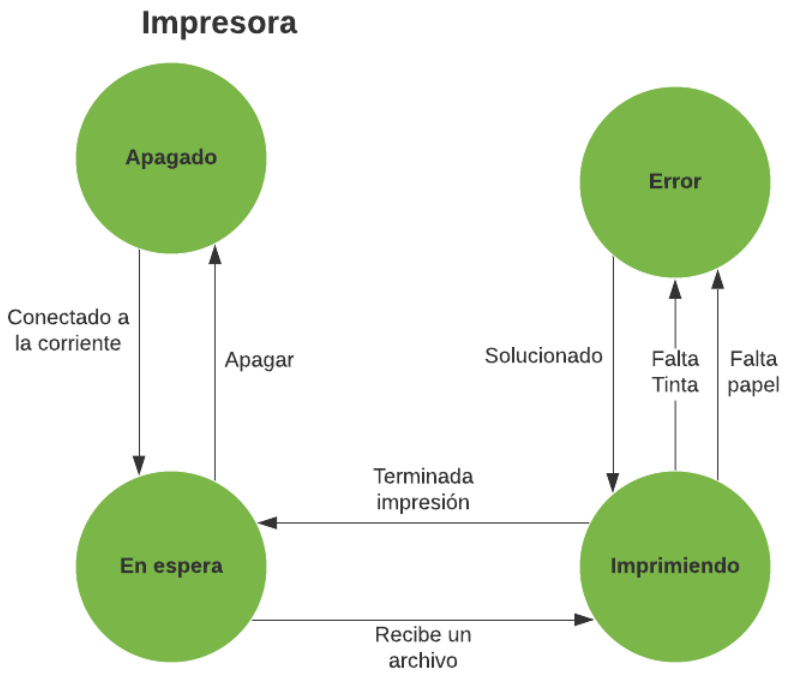
   Se puede modificar de la siguiente manera: cada vez que el robot entre en estado de “nueva búsqueda” calculamos la distancia con respecto a los objetos y escogemos el mas cercano, utilizando la formula de distancia en 2 dimensiones.

$$d = \sqrt{(X_2 - X_r)^2 + (Y_2 - Y_r)^2}$$

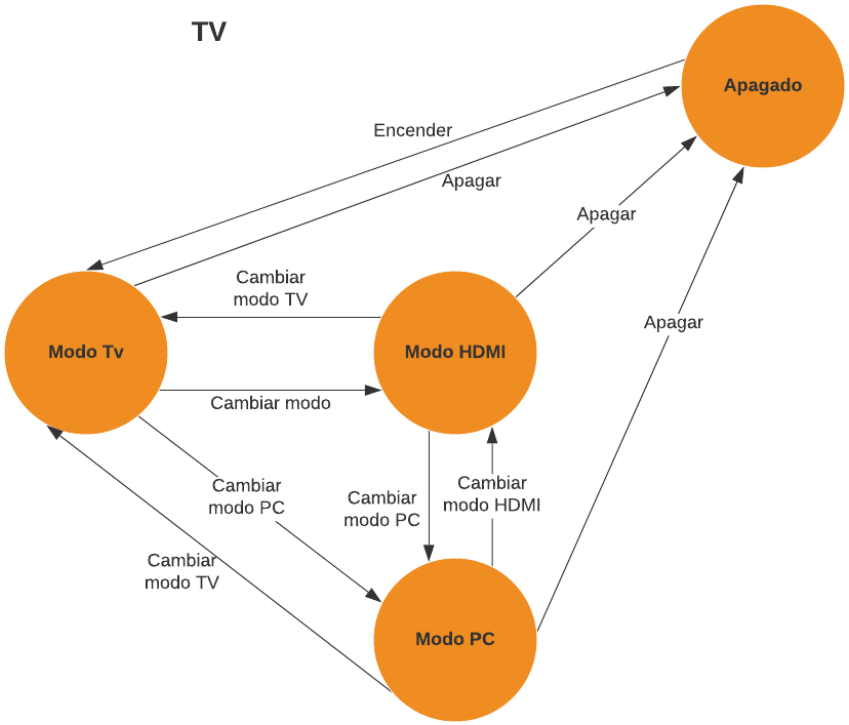
10. **¿Dónde se pueden aplicar las máquinas de estados finitos?**

    Las maquinas de estado finito pueden ser aplicadas en cualquier sistema que posea distintos estados y funciones de transición bien definidas.

11. Modelar una impresora como máquina de estados finitos.



12. Modelar un televisor como máquina de estados finitos.



Desarrollo del proyecto

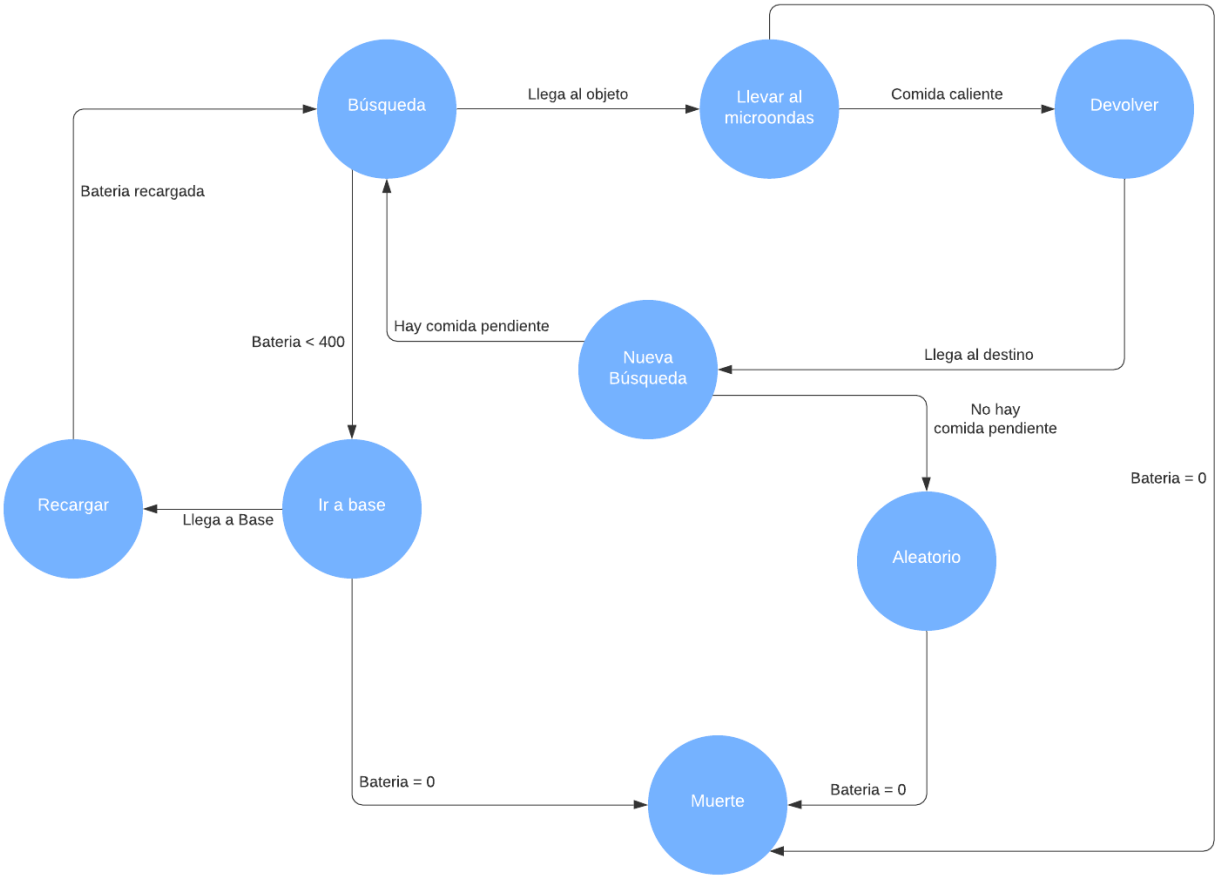


Fig. 1

El proyecto está orientado a una simulación de una simulación para un protocolo para calentar la comida en la Universidad Católica Andrés Bello (UCAB) a través del uso de un dron, realizando el recorrido que se muestra en el diagrama, representado en la figura 1, que cada lugar nuevo en donde se obtiene la comida, son ciertas zonas de recarga, que estarán ubicadas a lo largo de la universidad. Los estudiantes colocarán la comida en el dron, este se dirige a calentar la comida en algún microondas y se regresa.

Esto con el fin de agilizar el movimiento y flujo de los estudiantes a la hora de la comida, para acortar el tiempo en el que los estudiantes hacen la fila para calentar la comida y desplazarse con comodidad. A pesar de que, parece un juego es una simulación prototipo del proyecto que ciertos estudiantes emprendedores les encantaría implementar.

A continuación, se explica paso a paso el código del sistema:

```
1   using System;
2
3
4   namespace MEF
5   {
6       // Estructura usada para los objetos y la bateria
7       public struct S_objeto
8       {
9           public bool activo; // Indica si el objeto es visible o no
10          public int x,y;     // Coordenadas del objeto
11          public bool esperando; //Indica si esta en espera
12      }
13
14
15      /// <summary>
16      /// Esta clase representa a nuestra maquina de estados finitos.
17      /// </summary>
18      public class CMaquina
19      {
20          // Enumeracion de los diferentes estados
21          public enum  estados
22          {
23              BUSQUEDA,
24              LLEVARMICRO,
25              DEVOLVER,
26              NBUSQUEDA,
27              IRBATERIA,
28              RECARGAR,
29              MUERTO,
30              ALEATORIO,
31
32          };
```

```
33
34      // Esta variable representa el estado actual de la maquina
35      private int Estado;
36
37      // Estas variables son las coordenadas del robot
38      private int x,y;
39
40      // Arreglo para guardar una copia de los objetos
41      private S_objeto[] objetos = new S_objeto[10];
42      private S_objeto bateria;
43      private S_objeto estBase;
44
45      // Variable del indice del objeto que buscamos
46      private int indice;
47
48      // Variable para la energia;
49      private int energia;
50
51      // Creamos las propiedades necesarias
52      public int CoordX
53      {
54          get {return x;}
55      }
56
57      public int CoordY
58      {
59          get {return y;}
60      }
```

```
61
62          public int EstadoM
63          {
64              get {return Estado;}
65          }
66
67          public int EnergiaM
68          {
69              get { return energia; }
70          }
71
72          public CMaquina()
73          {
74              // Este es el contructor de la clase
75
76              // Inicializamos las variables
77
78              Estado=(int)estados.NBUSQUEDA;  // Colocamos el estado de inicio.
79              x=320;      // Coordenada X
80              y=240;      // Coordenada Y
81              indice=-1;  // Empezamos como si no hubiera objeto a buscar
82              energia=1000;
83          }
```

```
84
85          public void Inicializa(ref S_objeto [] Pobjetos, S_objeto Pbateria, S_objeto Pbase)
86          {
87              // Colocamos una copia de los objetos y la bateria
88              // para pode trabajar internamente con la informacion
89
90              objetos=Pobjetos;
91              bateria=Pbateria;
92              estBase = Pbase;
93
94          }
95
95
96          public void Control()
97          {
98              // Esta funcion controla la logica principal de la maquina de estados
99
100             switch(Estado)
101             {
102                 case (int)estados.BUSQUEDA:
103                     // Llevamos a cabo la accion del estado
104                     Busqueda();
105
106                     // Verificamos por transicion
107                     if(x==objetos[indice].x && y==objetos[indice].y)
108                     {
109                         // Desactivamos el objeto encontrado
110                         objetos[indice].activo=false;
111                         // Habilitamos espera
112                         objetos[indice].esperando = true;
113
114                         // Cambiamos de estado
115                         Estado =(int)estados.LLEVARMICRO;
116
117                     }
118
119                     if (energia < 400) // Checamos condicion de transicion
120                         Estado = (int)estados.IRBATERIA;
121
122                     break;
123
```

```
124
125             case (int)estados.LLEVARMICRO:
126                 // Llevamos a cabo la accion del estado
127                 LlevarComida();
128
129                 // Verificamos por transicion
130                 if (x == estBase.x && y == estBase.y)
131                     Estado = (int)estados.DEVOLVER;
132
133                 if (energia == 0)
134                     Estado = (int)estados.MUERTO;
135
136                 break;
137
138             case (int)estados.DEVOLVER:
139                 Devolver();
140
141                 if (x == objetos[indice].x && y == objetos[indice].y)
142                 {
143                     // Desactivamos el objeto encontrado
144                     objetos[indice].activo = false;
145                     // Fin de la espera
146                     objetos[indice].esperando = false;
147
148
149                     // Cambiamos de estado
150                     Estado = (int)estados.NBUSQUEDA;
151
```

```
152                 }
153                 break;
154 
155             case (int)estados.NBUSQUEDA:
156                 // Llevamos a cabo la accion del estado
157                 NuevaBusqueda();
158 
159                 // Verificamos por transicion
160                 if (indice==-1) // Si ya no hay objetos, entonces aleatorio
161                     Estado=(int)estados.ALEATORIO;
162                 else
163                     Estado=(int)estados.BUSQUEDA;
164 
165                 break;
166 
167             case (int)estados.IRBATERIA:
168                 // Llevamos a cabo la accion del estado
169                 IrBateria();
170 
171                 // Verificamos por transicion
172                 if(x==bateria.x && y==bateria.y)
173                     Estado=(int)estados.RECARGAR;
174 
175                 if(energia<=0)
176                     Estado=(int)estados.MUERTO;
177 
178                 break;
179 
180             case (int)estados.RECARGAR:
```

```
180                 case (int)estados.RECARGAR:
181                     // Llevamos a cabo la accion del estado
182                     Recargar();
183 
184                     // Hacemos la transicion
185                     Estado=(int)estados.BUSQUEDA;
186 
187                     break;
188 
189                 case (int)estados.MUERTO:
190                     // Llevamos a cabo la accion del estado
191                     Muerto();
192 
193                     // No hay condicion de transicion
194 
195                     break;
196 
197                 case (int)estados.ALEATORIO:
198                     // Llevamos a cabo la accion del estado
199                     Aleatorio();
200 
201                     // Verificamos por transicion
202                     if(energia==0)
203                         Estado=(int)estados.MUERTO;
204 
205                     break;
206 
207             }
208 
209         }
```

```
210
211         public void Busqueda()
212         {
213             // En esta funcion colocamos la logica del estado Busqueda
214
215             // Nos dirigimos hacia el objeto actual
216             if(x<objetos[indice].x)
217                 x++;
218             else if(x>objetos[indice].x)
219                 x--;
220
221             if(y<objetos[indice].y)
222                 y++;
223             else if(y>objetos[indice].y)
224                 y --;
225
226             // Disminuimos la energia
227             energia--;
228
229         }
230
231          public void LlevarComida()
232          {
233              //Nos dirigimos hacia la base
234
235              if (x < estBase.x)
236                  x ++;
237              else if (x > estBase.x)
238                  x --;
239
240              if (y < estBase.y)
241                  y ++;
242              else if (y > estBase.y)
243                  y --;
244
245              //Consumo
246              energia--;
247          }
248
```

```
249     public void Devolver() {
250         Busqueda();
251     }
252 
253     public void NuevaBusqueda()
254     {
255         // En esta funcion colocamos la logica del estado Nueva Busqueda
256         // Verificamos que haya otro objeto a buscar
257         indice=-1;
258 
259         // Recorremos el arreglo buscando algun objeto activo
260         for(int n=0;n<10;n++)
261         {
262             if(objetos[n].activo==true)
263                 indice=n;
264         }
265     }
266 
```

```
267      public void Aleatorio()
268      {
269          // En esta funcion colocamos la logica del estado Aleatorio
270          // Se mueve al azar
271 
272          // Cremos un objeto para tener valores aleatorios
273          Random random=new Random();
274 
275          int nx=random.Next(0,3);
276          int ny=random.Next(0,3);
277 
278          // Modificamos la posicion al azar
279          x+=nx-1;
280          y+=ny-1;
281 
282          energia--;
283 
284      }
285 
```

```
286         public void IrBateria()
287         {
288             // En esta funcion colocamos la logica del estado Ir Bateria
289 
290             // Nos dirigimos hacia la bateria
291             if(x<bateria.x)
292                 x++;
293             else if(x>bateria.x)
294                 x --;
295 
296             if (y < bateria.y)
297                 y++;
298             else if (y > bateria.y)
299                 y--;
300 
301             // Disminuimos la energia
302             energia--;
303 
304         }
305 
306         public void Recargar()
307         {
308             // En esta funcion colocamos la logica del estado Recargar
309             energia=1500;
310 
311         }
312 
313          public void Muerto()
314          {
315              // En esta funcion colocamos la logica del estado Muerto
316 
317              // Sonamos un beep de la computadora
318 
319 
320          }
321 
322 
323 
324     }
325 }
326 
```

Cada parte del código está con comentarios (antecedidos por //) la explicación de cada procedimiento y/o función utilizada, la creación de los estados y las condiciones de transición.

A continuación, una recopilación del funcionamiento visual del código anterior agregado con una parte visual que oculta la abstracción de la máquina de estados finitos, realiza el trabajo demostrado en el diagrama de estados:

Se puede observar la posición inicial de la máquina de estado:

Nueva búsqueda:

Llevando al microondas:

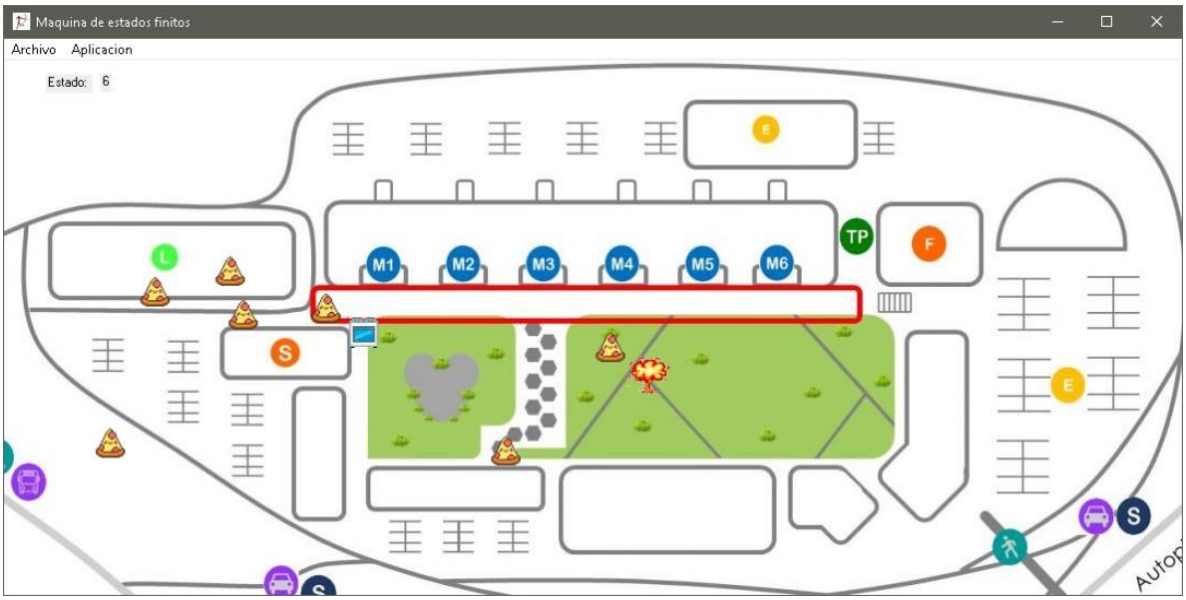
Calentando comida:

Devolviendo comida calentada:

Iniciando nueva búsqueda:

Recargando energía:

Por último, se ve que el estado de muerte visualiza con una imagen de una explosión.



Como se mencionó a lo largo del informe, el potencial que poseen las máquinas de estados finitos es ilimitado, el desarrollo de cualquier sistema puede ser basado en el comportamiento lógico de estas, por lo cual da pie a que todo se realice con cada vez menos abstracción.

Las simulaciones pueden ser modeladas de diferentes formas, a través de modelos matemáticos, diagramas de estado, entre otros, ya que estas son abstracciones de lo que percibimos y se intenta que esta sea más acercada a la realidad en sí, por lo que posee muchas variantes, a lo cual a veces no se está conscientes de todas. Es importante aclarar que las representaciones de con máquinas de estados finitos, son representaciones bastante abstractas y dependiendo del modelo pueden ser más o menos complejas.