Máquinas de Estado Finito

Introducción

Las máquinas de estado finito son una herramienta muy útil para especificar aspectos relacionados con tiempo real, dominios reactivos o autónomos, computación reactiva, protocolos, circuitos, arquitecturas de software, etc.

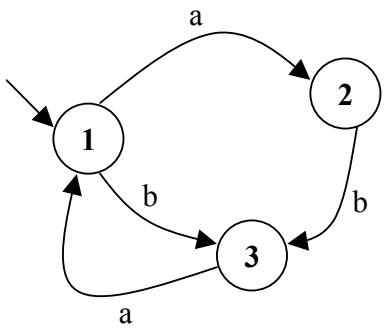
El modelo de FSM (Finite State Machine) es un modelo que posee sintaxis y semántica formales y que sirve para representar aspectos dinámicos que no se expresan en otros diagramas.

Sintaxis

Las máquinas de estado finito se definen como una tupla $\langle S, \Sigma, A \subseteq S \times \Sigma \times S, sk \rangle$, donde:

- $S = \{s_1, s_2, ..., s_m\}$: es un conjunto finito de nodos.
- Σ: es un alfabeto finito de etiquetas.
- A: es un conjunto finito de aristas etiquetadas que unen nodos.
- $sk \in S$: es el estado inicial.

Ejemplo



- $\langle S = \{1,2,3\}$,
- $\Sigma = \{a,b\}$,
- $A = \{(1,a,2),(2,b,3),(3,a,1),(1,b,3)\}$,
- $sk = 1 \rangle$

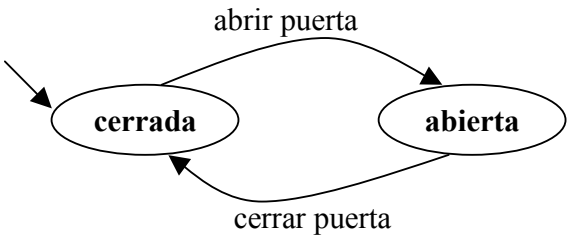
Vale la pena destacar que **formalmente** la máquina de estado es la tupla anterior y no el "dibujo". Este es tan sólo una representación gráfica de la máquina de estado para tener una más sencilla y rápida visualización de su contenido.

Semántica

Los nodos representan los posibles estados de aquello que se desea modelar. Las etiquetas representan eventos que provocan un cambio. Las aristas determinan de qué manera cada estado, dado un evento, deriva en otro estado.

Ejemplo

Supongamos que se quiere modelar el comportamiento de una puerta. La puerta, inicialmente cerrada, puede pasar a estar abierta tras el evento "abrir puerta". Una vez abierta, puede pasar al estado cerrada, tras el evento "cerrar puerta".

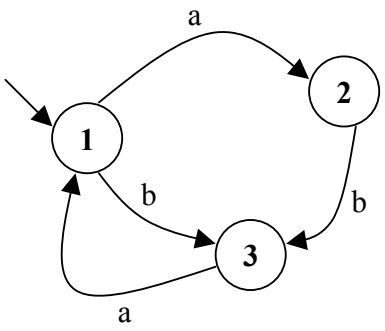


Trazas

El conjunto de posibles trazas correspondientes a una máquina de estado finitos, se puede definir en término de grafos, cómo el conjunto de todos los caminos (de ejes) alcanzables desde el estado inicial.

Ejemplo

Dada la FSM de ejemplo:



Las trazas de esta FSM son:

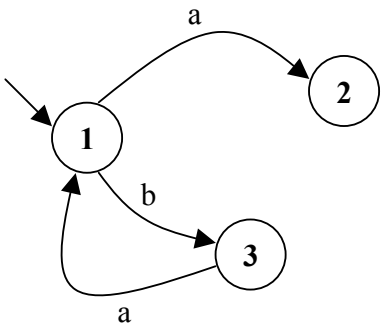
- {a, b, a} correspondiente a 1, 2, 3, 1
- {b, a} correspondiente a 1, 3, 1
- {a, b, a, b, a} correspondiente a 1, 2, 3, 1, 3, 1
- {b, a, a, b, a} correspondiente a 1, 3, 1, 2, 3, 1
- {b, a, b, a, ... , b, a} 1, 3, 1, 3, …, 1, 3
- Etc…

Deadlock

Formalmente hablando, una FSM $\langle S, \Sigma, A \subseteq S \times \Sigma \times S, sk \rangle$, tiene deadlock, si existe algún nodo $s \in S$, tal que no existen un evento e y un nodo $t \in S$ tal que $(s, e, t) \in A$. En otras palabras, si existe algún nodo que no posea "salida" para ningún evento.

Ejemplo

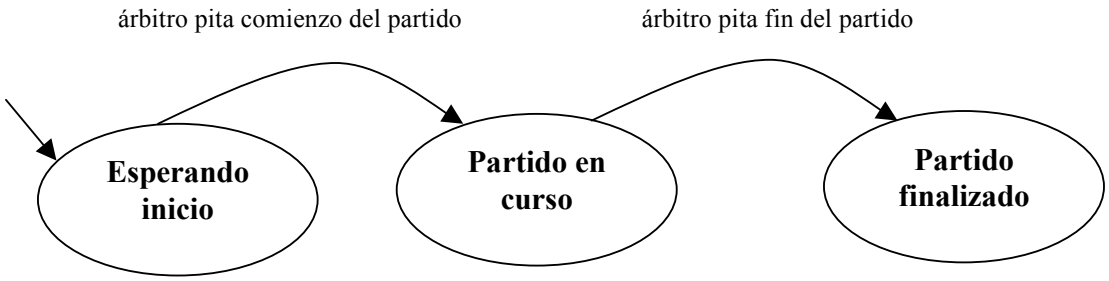
El estado 2 no posee salida alguna.



Semántica del deadlock

La existencia de deadlock no necesariamente habla de un mal modelado. Más adelante veremos que puede ser un efecto no deseado de la composición de FSMs.

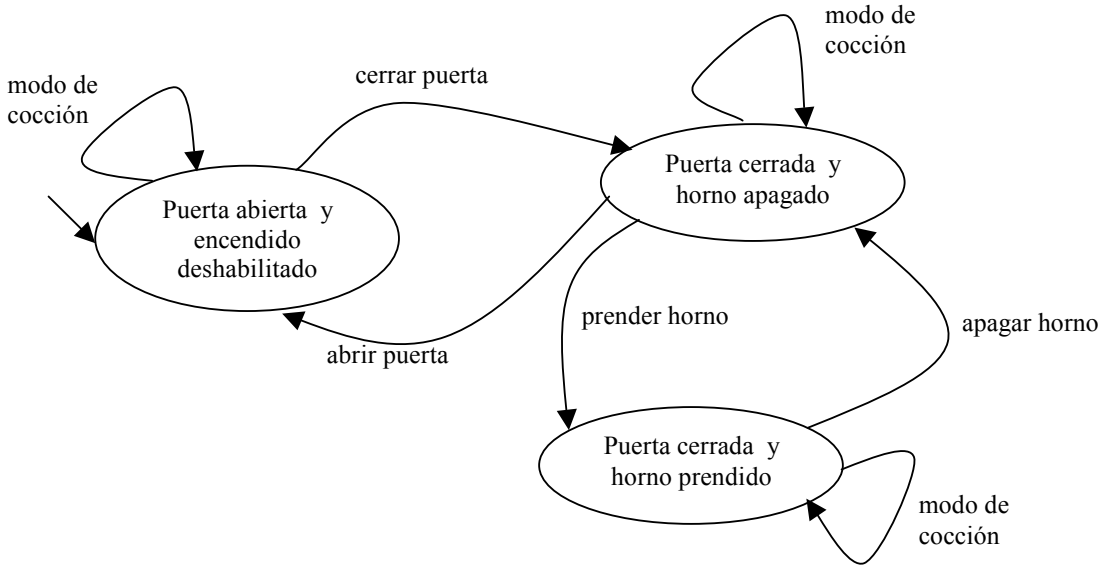
Un ejemplo en donde el deadlock no necesariamente "está mal", podría ser el siguiente: Queremos modelar un partido de fútbol. Un partido de fútbol, en particular, se organiza, se juega y finaliza. Y una vez finalizado, ese partido no se juega ni organiza nunca más. Entonces podríamos modelarlo de la siguiente manera:



Composición de FSM

Supongamos que se quiere modelar el siguiente comportamiento de un horno microondas: El horno microondas posee una puerta. Si la puerta está cerrada, entonces puede estar o no en funcionamiento (según se prenda o apague). Estando prendido no es posible abrir la puerta del horno sin antes apagarlo.

También asumamos lo siguiente: en cualquier momento es posible establecer el modo de cocción.



No es obligatorio etiquetar los nodos, sin embargo puede resultar muy útil hacerlo: sobre todo aquellos nodos que representan los estados principales de que se esté modelando.

Muchas veces es posible descomponer un complicado problema en problemas más pequeños y más fáciles de tratar. Esto también es aplicable a las FSMs: una FSM puede descomponerse en varias FSMs más simples. Sin embargo (como verán resolviendo los ejercicios de la práctica) no es trivial hacerlo, y hay que tener mucho cuidado de que la composición de las maquinas sencillas represente exactamente a la FSM descompuesta y no a una “parecida”.

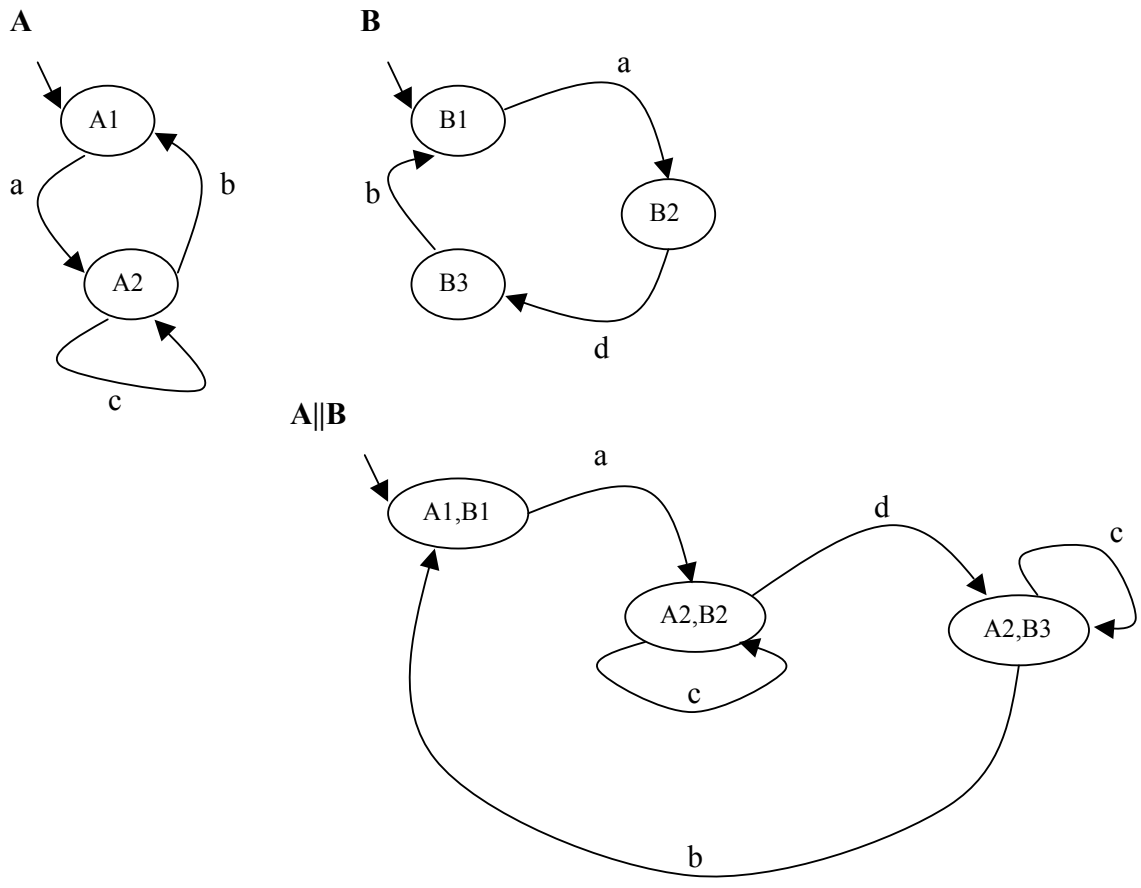
Sintaxis de la composición

La composición de FSMs se realiza a través de la sincronización de etiquetas.

Sean A = $\langle S_A, \Sigma_A, A_A \subseteq S_A \times \Sigma_A \times S_A, sk_A \rangle$ y B = $\langle S_B, \Sigma_B, A_B \subseteq S_B \times \Sigma_B \times S_B, sk_B \rangle$ dos FSM.
La composición paralela A || B se define como
$\langle S, \Sigma_A \cup \Sigma_B, A \subseteq (S_A \times S_B) \times (\Sigma_A \cup \Sigma_B) \times (S_A \times S_B), (sk_A, sk_B) \rangle$ donde:

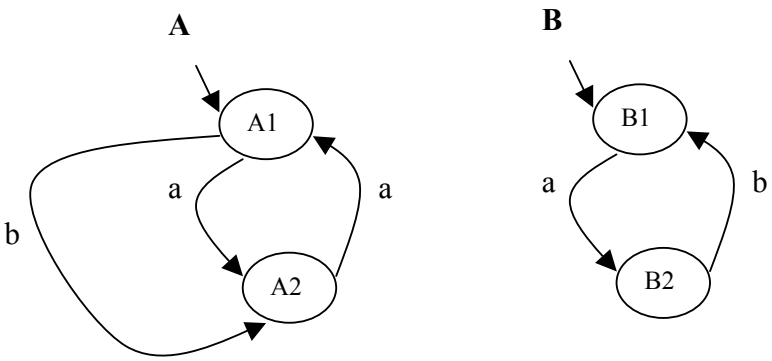
- $S \subseteq S_A \times S_B$
- $((s_a, s_b), e, (s_c, s_d)) \in A$ sólo si se cumple alguna de estas condiciones:
  - $(s_a, e, s_c) \in A_A \wedge e \notin \Sigma_B \wedge s_b = s_d$
  - $(s_b, e, s_d) \in A_B \wedge e \notin \Sigma_A \wedge s_a = s_c$
  - $(s_a, e, s_c) \in A_A \wedge (s_b, e, s_d) \in A_B \wedge e \in \Sigma_A \cap \Sigma_B$

Ejemplo

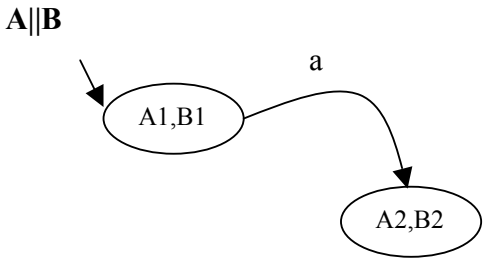


Sobre la composición

Analicemos el siguiente ejemplo:



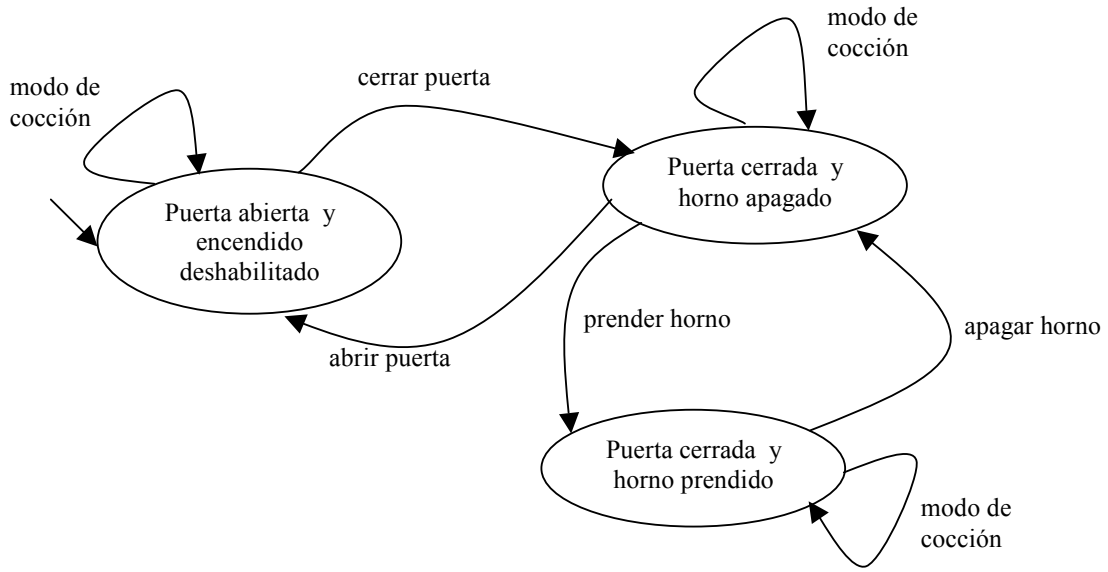
¿Qué sucede con **A||B**?



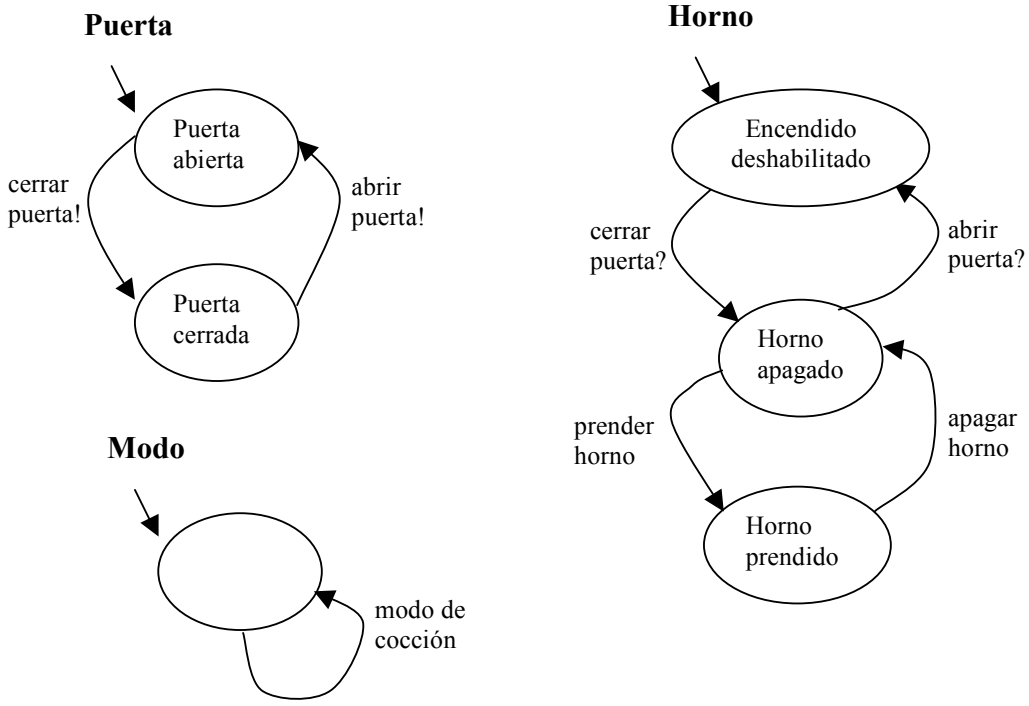
Según nuestras reglas de composición, A||B quedaría con el estado inicial (A1,B1) y la única transición posible sería a, dejando a la máquina en el estado (A2, B2) y en deadlock. Las máquinas A y B no poseían deadlock, pero su composición sí. También podría pasar al revés. Componer dos FSM no es simplemente poner el símbolo || entre sus nombres: se debe analizar el posible resultado para comprobar que se está modelando lo que uno realmente desea modelar.

Ejemplo

Volviendo a nuestro ejemplo:



Podemos descomponer este problema en tres partes: por un lado modelaremos una puerta que se abre y se cierra, y por otro un horno que se prende y apaga y trataremos el modo de cocción de manera independiente.



HornoMicroondas = Puerta || Horno || Modo

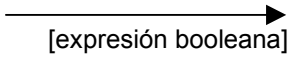
Cuando se hagan composiciones de FSMs, en las transiciones que posean etiquetas con el mismo nombre en dos o más maquinas de estado diferentes (es decir las transiciones que componen) se utilizará el símbolo de exclamación y de interrogación para diferenciar entre el arco que realmente dispara la transición, del que recibe el impulso. Al nombre de etiqueta del primero se le concatenará un signo de exclamación, mientras que al segundo uno de exclamación. La semántica es tan simple como que la transición con el signo de exclamación es la que realmente emite y dispara el evento, y las que poseen el de interrogación son las que lo reciben.

Extensiones

Para aumentar el poder expresivo, existen extensiones al lenguaje de FSM visto anteriormente.

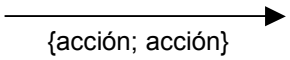
Condiciones

Son expresiones booleanas que determinan si una transición entre estados puede realizarse. Se escriben entre corchetes.



Acciones

Inmediatamente después de ejecutar una transición y antes de arribar al siguiente estado, la máquina de estado ejecuta una acción. Básicamente, para nosotros, las acciones serán asignaciones de valores a variables. En algunos contextos, dentro de las acciones, también se lanzan eventos (para luego sincronizar otras máquinas). Se escriben entre corchetes.



Variables

Sus valores posibles deben pertenecer a un dominio finito. Pueden utilizarse estructuras como arreglos o tipos abstractos de datos en general (cómo por ejemplo conjuntos o pilas).

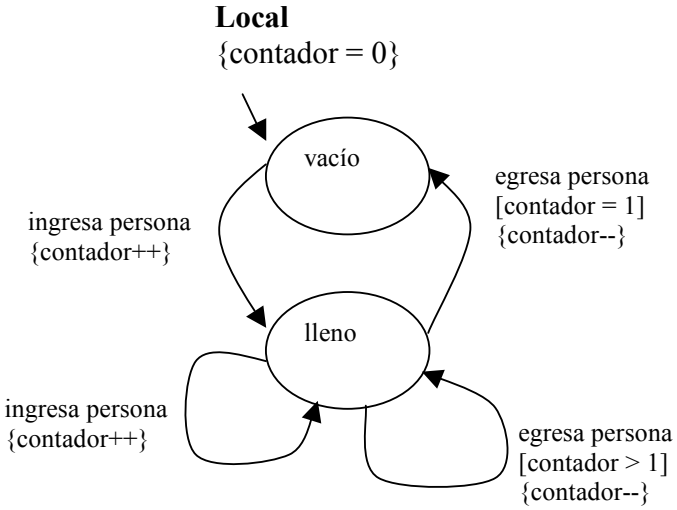
Los variables pueden usarse dentro de las expresiones booleanas de las condiciones y su valor puede cambiarse utilizando acciones, generalmente asignaciones. A su vez, salvo explicitado lo contrario, las variables son compartidas por todas las FSMs de la composición (es decir pueden verse como globales a todas las máquinas de estado).

Ejemplo

Un local de ropa puede estar vacío o lleno según la cantidad de personas que hay en su interior. Inicialmente está vacío, pero luego de ingresar la primer persona esta lleno. A partir de ahí, la gente puede ingresar o salir. Cuando sale el último, vuelve a estar vacío.

Todas las variables se deben definir de antemano:

contador : [0, 99999999] (por simplicidad, permitiremos decir $\mathbb{N}$. Sin embargo, debemos resaltar que esto no es del todo correcto: el dominio debe tener un número finito de valores).

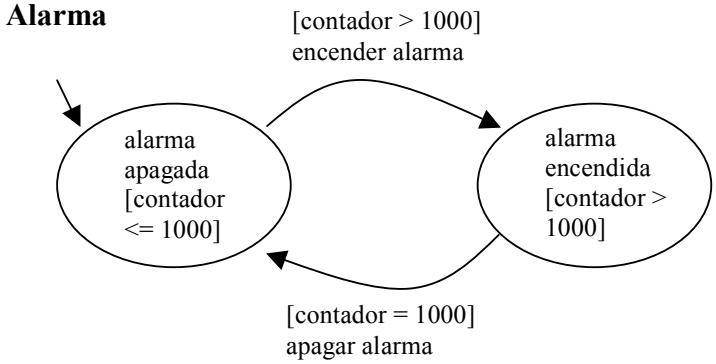


Condiciones de salida de un estado

Es posible colocar condiciones booleanas dentro de los estados (utilizando [expresión booleana] dentro de los nodos) para obligar a que se salga del mismo. Mientras la condición agregada a un estado sea verdadera es posible mantenerse en dicho estado o salir del mismo a través de una transición cualquiera. Sin embargo apenas dicha condición se haga falsa la maquina estará obligada a dejar dicho estado por alguna de las transiciones de salida. Si hay muchas posibles se saldrá por cualquiera de estas de forma no determinística. Si en cambio no hay ninguna transición posible, se caerá indefectiblemente en un deadlock.

Supongamos que al ejemplo anterior del local de ropa se le agrega la condición de que si hay más de 1000 personas dentro debe sonar una alarma. Una forma de representar dicha situación es componiendo la siguiente FSM a la ya hecha:

Ejemplo



LocalSeguro = Local || Alarma

FSMs Temporizadas

Muchas veces los disparadores de los eventos que provocan los cambios de estado de las máquinas son iniciados por el paso del tiempo. Por ej. el cambio de luz de los semáforos, un rociador automático de césped que se enciende y riega cada 2 horas por 10 minutos, o un sistema de dosificación de químicos en una cañería que cada determinada cantidad de tiempo inyecta distintas sustancias en un orden especial. En todos estos casos la decisión de que ocurran los eventos es disparada por el tiempo. Para poder representar este funcionamiento utilizando FSMs se hace necesario extenderlas agregando un componente extra denominado "timer".

Este será declarado junto con el resto de las variables de la siguiente manera:

nombreTimer: timer. En -unidad de tiempo-. (por ej. segundos).

Las únicas operaciones permitidas por el timer es "resetearlo" (es decir colocar su cuenta en 0) y chequear su valor (es decir, ver el tiempo transcurrido desde su último "reset"). Una vez que el timer es "reseteado" empieza a contar el tiempo transcurrido hacia adelante, y eso es todo lo que sabe hacer (no se lo puede setear en un valor determinado por ej.). Podremos chequear el tiempo transcurrido a través del uso de condiciones y podremos resetearlo utilizando acciones.

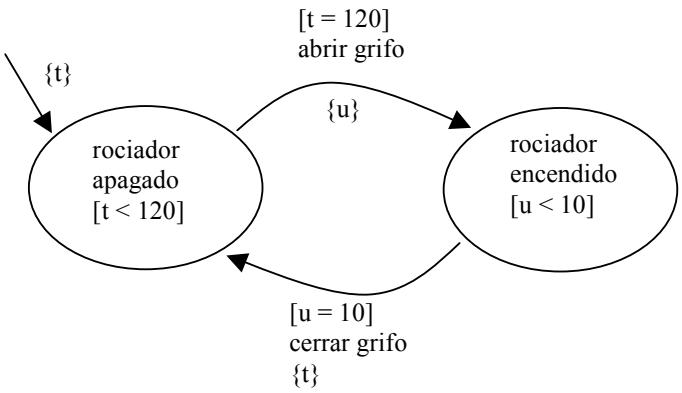
La sintaxis para resetear el timer es colocando al nombre del mismo dentro de llaves.

Plasmemos el ejemplo del rociador automático en una FSM para ver como se puede utilizar esta extensión para representar dicha situación:

Ejemplo

t : timer. En minutos.
u: timer. En minutos.

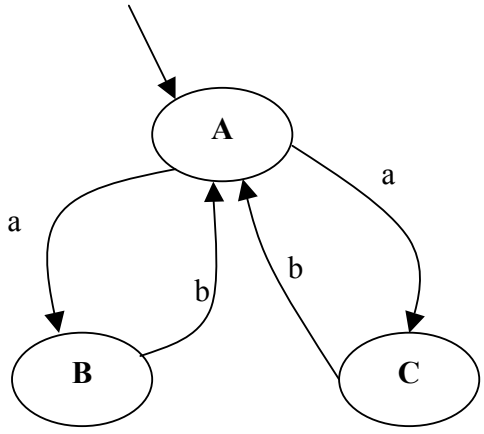
Rociador



No Determinismo

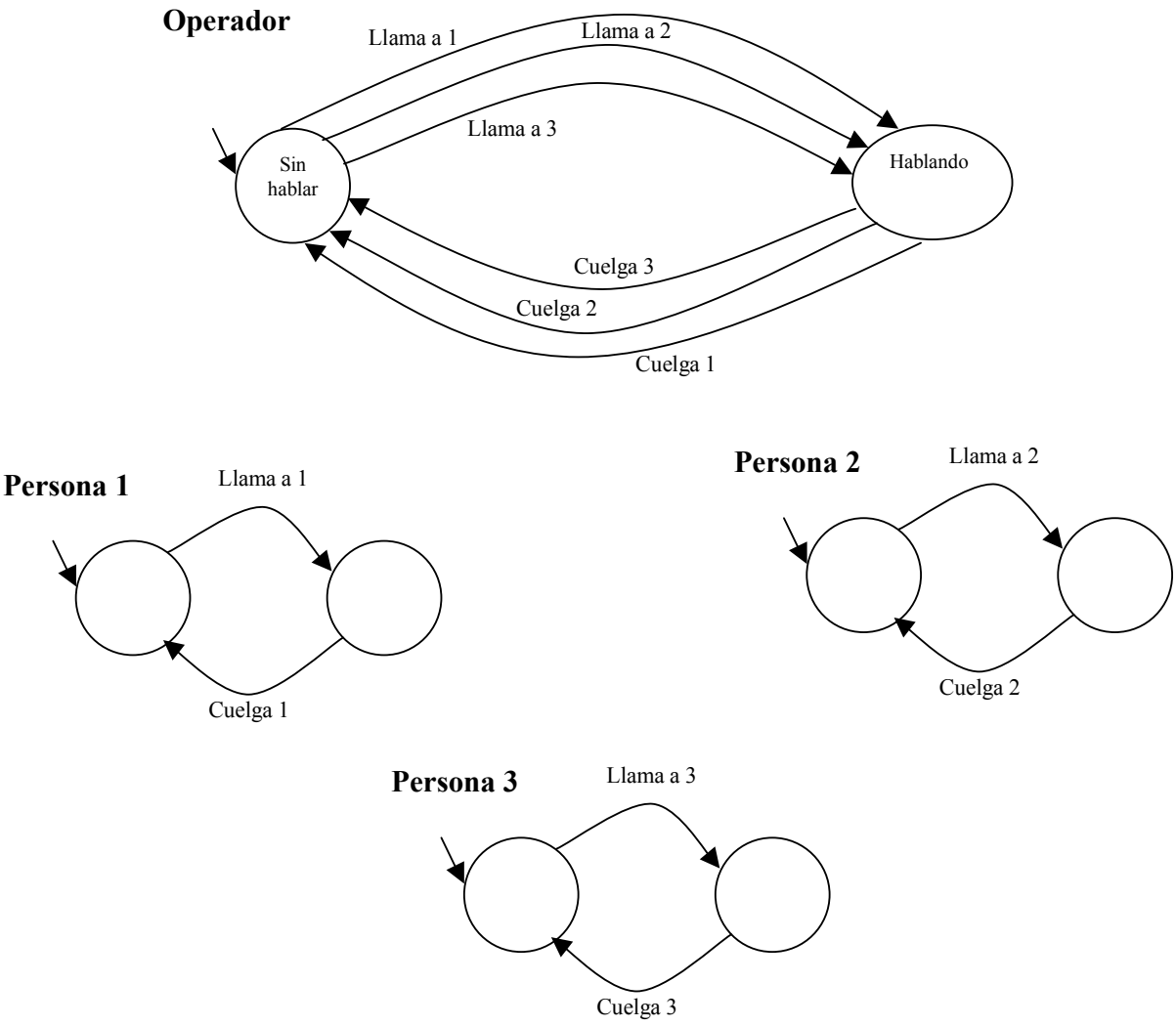
Si dado un estado y dado un evento, existe más de una transición válida (cuya condición, si existe, se satisface), hacia distintos estados, se asumirá un comportamiento no determinístico: es decir, cualquiera de los estados posibles podrá ser alcanzado, sin orden alguno.

Ejemplo



Abusos de Notación

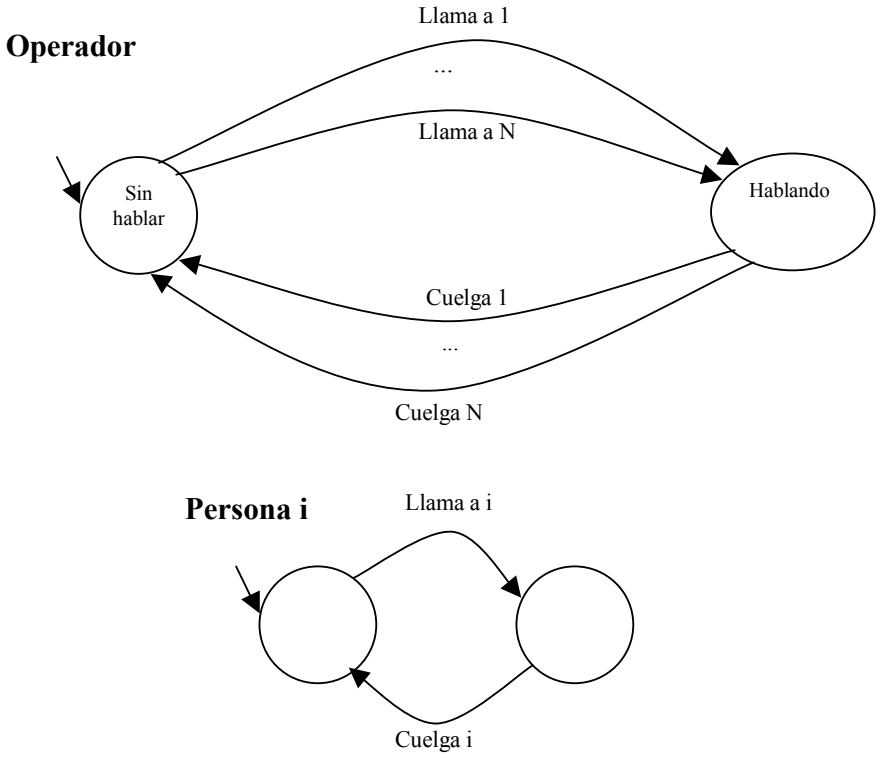
Supongamos que se quiere modelar lo siguiente: un operador telefónico puede llamar a tres personas diferentes, pero sólo a una de estas a la vez.



Operador || Persona 1 || Persona 2 || Persona 3

Si en lugar de tres personas, las mismas son 100, o 1000, o 10000, rapidamente nos encontraremos con un problema.

Para modelar lo anterior, aclarando que la cantidad de personas es finita, se podrá utilizar la siguiente notación:



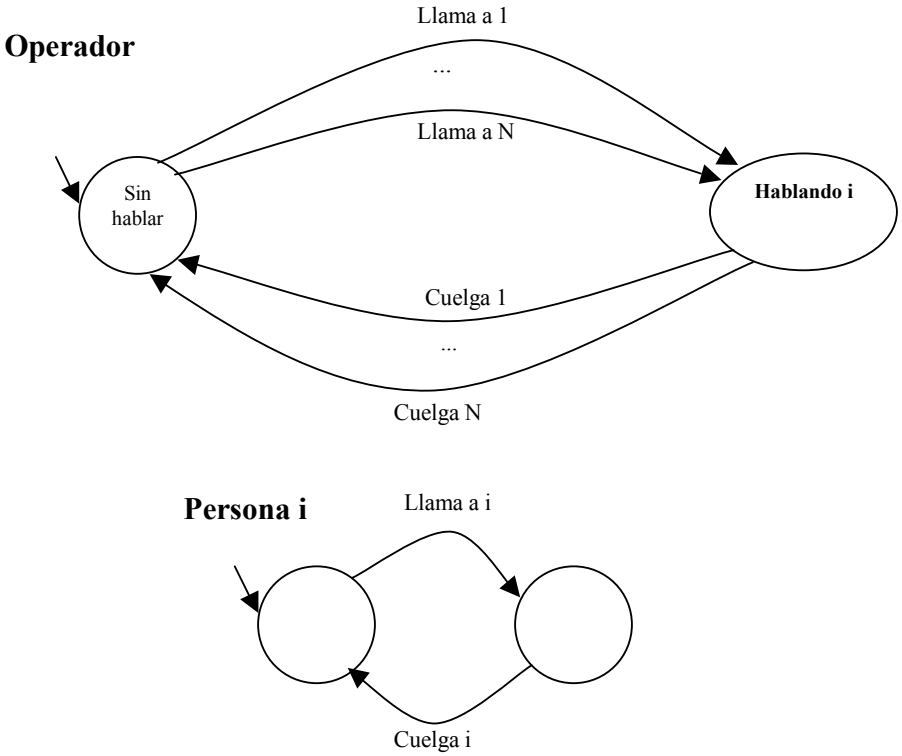
Operador || Persona 1 || ... || Persona N

Es importante aclarar que al decir *Persona i* o *Llama a i*, la letra i **no es una variable**. Debe pensarse como una macro que se reemplaza por el número correspondiente. *Persona i* es el nombre de la máquina. *Llama a i* es una etiqueta, su nombre no puede ser dinámico.

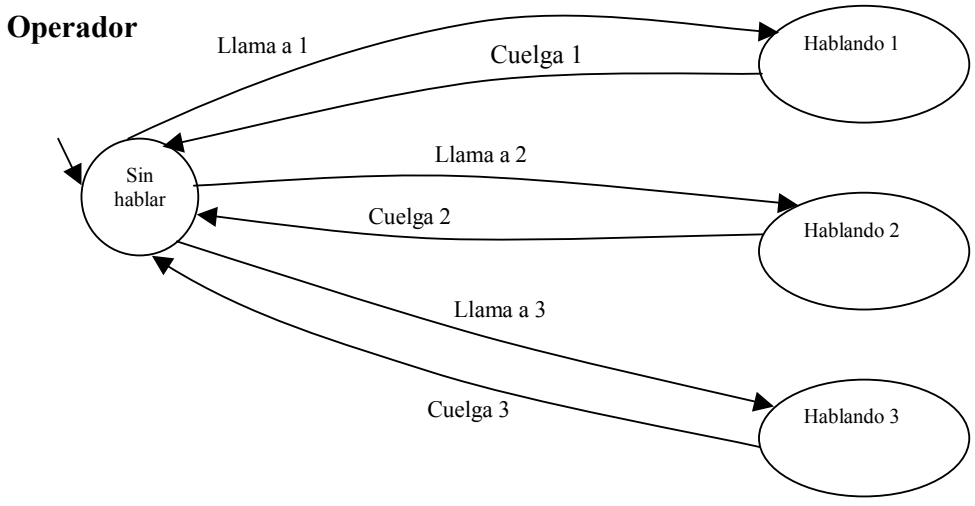
En el caso del operador, a pesar de que responde a muchos *Llama a i* posee sólo dos estados: Sin hablar o Hablando.

Por simplicidad, no se recomienda "indizar los estados". Es decir, hablar de por ejemplo del estado Hablando i. La razón de esto es que facilmente se pierde control de lo que se está queriendo modelar.

El problema anterior se podría pensar de la siguiente manera:



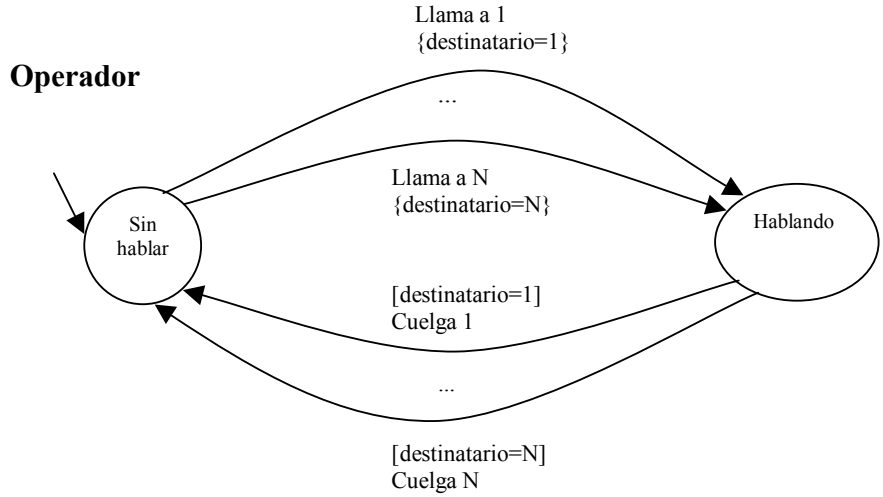
Es decir, identificar en el operador, no sólo si está hablando o no, sino que también con quien lo está haciendo (en el otro ejemplo, esta información es deducible).



Puede verse cómo, al explotar los estados, se forma una especie de abanico. Imaginense ahora que desde el estado ***Hablando i***, sale otra flecha cuyo evento también se encuentra "indizado" y que llega a otro estado también "indizado". Este tipo de abuso, oculta mucha información que hace a la interpretación de la máquina de estados.

Otra forma de hacerlo (probablemente no la más bonita desde el punto de vista de la claridad expositiva pero sintácticamente correcta) es la siguiente, ahora utilizando una única maquina de estados pero agregando variables y condiciones:

destinatario: [1, N]



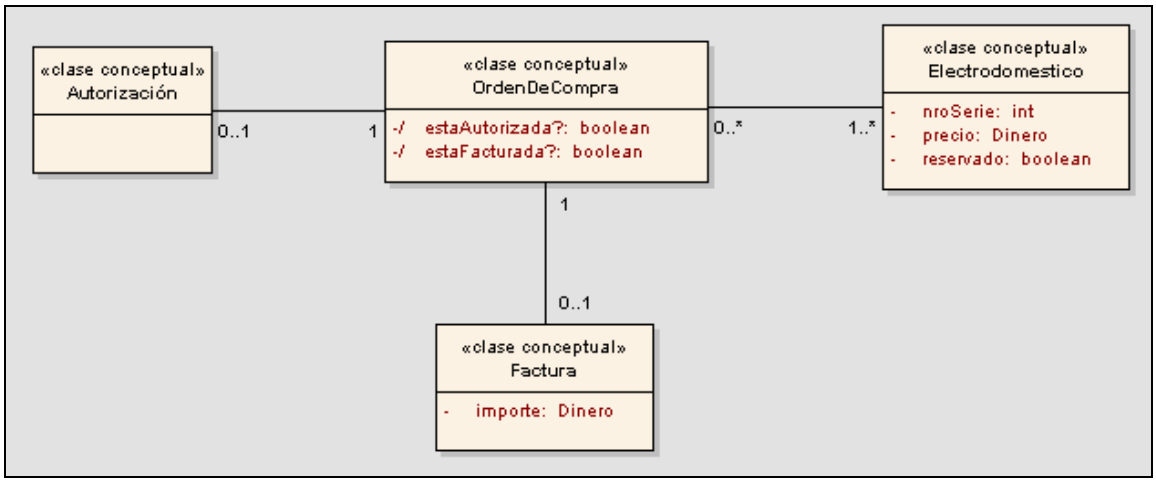
Es decir se reemplazan las N FSMs "Persona i" por una variable "destinatario" que almacena al receptor de la llamada del operador y restringe la salida del estado "Hablando" por la única transición correcta.

Modelo Conceptual

Las máquinas de estado finito, puede utilizarse para especificar aspectos dinámicos de problemas. Dichos problemas, probablemente también posean aspectos estáticos: estructuras de datos necesarias, entidades relacionadas entre sí, invariantes, etc. Muchos de estos aspectos pueden modelarse a través de un modelo conceptual. A veces, puede ser útil especificar el comportamiento de las entidades identificadas en el modelo conceptual: sus posibles cambios de estados. En estos casos, es interesante combinar distintas técnicas: modelo conceptual, OCL y FSM.

El objetivo será explicar la semántica de las transiciones de la FSM, identificando responsabilidades en las entidades, las cuales podrán ser especificadas formalmente utilizando OCL.

Ejemplo



Supongamos el siguiente dominio de problema: al ingresar una orden de compra, se determinan cuáles son los electrodomésticos que se venderán. Luego, cuando la orden es autorizada, los electrodomésticos son reservados. Sólo después de que la orden se encuentra autorizada, ésta puede ser facturada. El importe de la factura debe ser igual a la suma de los precios de los electrodomésticos.

// Los electrodomésticos no deben estar reservados si la orden aún no fue autorizada.
Context: OrdenDeCompra
Inv: self.estaAutorizada? = False implies (self.electrodomesticos->forAll(e | e.reservado = false))

// Una orden está autorizada, si posee una autorización asociada.
Context: OrdenDeCompra :: estaAutorizada? : boolean
Derive: self.autorización->size() > 0

// Una orden está facturada, si posee una factura asociada.
Context: OrdenDeCompra :: estaFacturada? : boolean
Derive: self.factura->size() > 0

// Una orden no puede ser facturada sino fue autorizada previamente.
Context: OrdenDeCompra
Inv: self.estaAutorizada? = False implies (self.estaFacturada? = False)

// El importe de la factura debe ser igual a la suma de los precios de los electrodomésticos.
Context OrdenDeCompra
Inv: self.estaFacturada? implies (self.factura.importe = self.electrodomesticos.precio->sum())

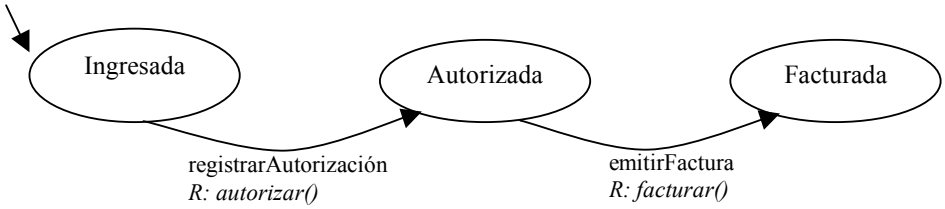
El ciclo de vida de una orden de compra, es el siguiente:

OrdenDeCompra

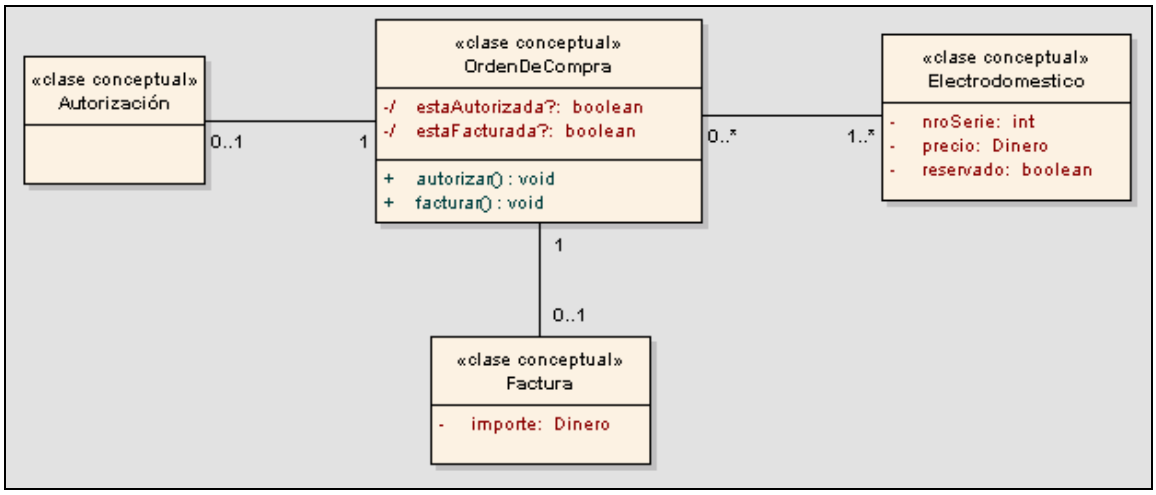
Cuando una FSM describa el comportamiento de una sola entidad, podremos fijar el contexto de la misma y podremos delegar las acciones correspondientes a algunos eventos, en responsabilidades de la entidad contexto.

Context: Orden de Compra

OrdenDeCompra



Y luego se deberá refinar el modelo conceptual:



Y con OCL especificar las responsabilidades:

Context: OrdenDeCompra:: autorizar
// No debe estar autorizada
Pre: self.estaAutorizada = False
// Luego de ejecutar esta responsabilidad, se crea una autorización asociada a la orden. Y se reservan todos los electrodomesticos de la orden.
Post: self.autorización->size() = 1 and self.autorizacion.oclIsNew() and
self.electrodomesticos->forAll(e | e.reservado = true)

Context: OrdenDeCompra:: facturar
// Debe estar autorizada y no debe estar facturada
Pre: self.estaAutorizada = True and self.estaFacturada = False
// Luego de ejecutar esta responsabilidad, se crea una factura asociada a la orden.
Post: self.factura->size() = 1 and self.factura.oclIsNew() and
self.factura.importe = self.electrodomesticos.precio->sum()

Aclaraciones

- La incorporación de estas extensiones, no condiciona ni cambia el comportamiento de la FSM ni su posible composición. El objetivo sólo es adicionar información semántica sobre qué significa, con respecto al modelo conceptual, que cierta entidad cambie de estado.

- Una vez establecido el contexto de una FSM, es posible utilizar expresiones OCL en las condiciones y acciones: se cuenta con los valores de los atributos (accediendo mediante el operador *self*) y con toda la información disponible mediante la navegación del modelo conceptual.

Ejercicio Resuelto

Cierta estancia cuenta con tres parcelas en las cuales realiza sus cultivos. Con el fin de optimizar su productividad, se ha definido una política con respecto a cómo trabajar la tierra.

La primer etapa consiste en preparar la tierra en todas las parcelas. Para ello se utiliza un tractor. La estancia posee un único tractor, el cual puede ser usado de a una parcela a la vez (y luego liberado). No hay un orden predeterminado para estas actividades, es más, todas las parcelas "se pelean" entre sí para el uso del tractor. De esta manera, mientras que en una parcela se está realizando el preparado de la tierra, las demás parcelas deben esperar a que termine.

Recién cuando la tierra de todas las parcelas ha sido preparada se da inicio a la segunda etapa.

Durante la segunda etapa se siembran ***simultáneamente*** todas las parcelas (no se utiliza el tractor para tal tarea). La manera de hacer el sembrado no es relevante (por lo general se utilizan aviones especialmente equipados).

La tercer etapa es comprendida por la cosecha de cada parcela, nuevamente utilizando el único tractor (con las mismas condiciones en cuanto a su uso con respecto al preparado de la tierra). Luego de realizar la cosecha de todas las parcelas, se está en condiciones de dar inició, nuevamente, a la primer etapa.

1. Modele la política implementada por la estancia, utilizando FSM.
2. ¿Cómo extendería el modelo para contemplar n parcelas?

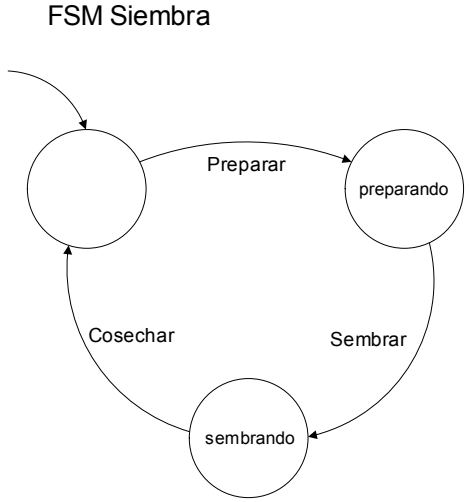
Respuesta sugerida

Vamos a resolver el problema para n parcelas directamente.
En este caso N sería 3.

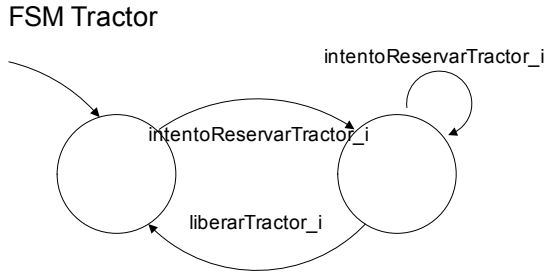
La respuesta es la composición paralela de las máquinas que describimos a continuación, esto se simboliza:

Siembra || Tractor || Parcela1 || .. || ParcelaN

Modelamos con una máquina la secuencia de operaciones que se realizarán en las parcelas:



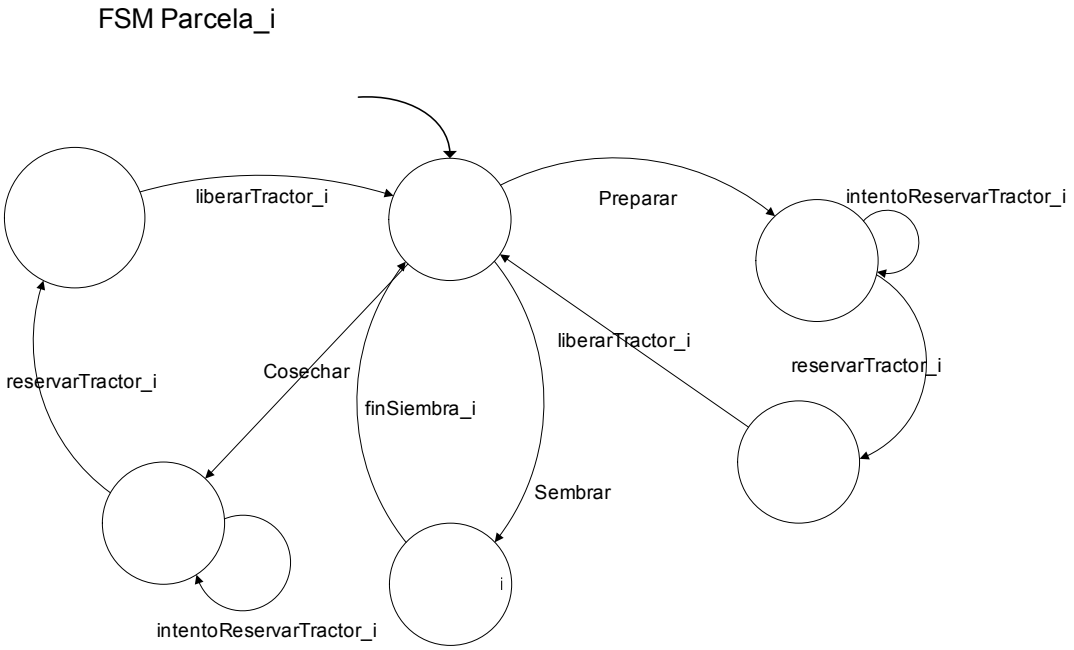
Luego modelamos el uso del tractor con otra máquina:



NOTA: Por cada evento hay n transiciones, una pc
cada parcela.

La inclusión de la transición intentoReservarTractor_i se hace para modelar que en realidad al no poder obtener el tractor cada parcela sigue intentando obtenerlo.

Finalmente tenemos la máquina que modela cada parcela:



Otra solución podría ser la siguiente

Siembra || Parcela_1 || …. || Parcela_N

En este caso no modelamos al uso del tractor con una máquina, sino que utilizamos una variable que nos dice si ya esta reservado o no.