Práctica 4: Procesos de negocio

Especificación de procesos de negocio

Un proceso de negocio describe, en una notación muy básica, un conjunto de tareas que describen un proceso. Aunque hay complejas notaciones como BPMN o BPEL con una gran variedad de operadores, tipos de tareas, etc., en esta práctica vamos a utilizar una notación muy sencilla (subconjunto de BPMN) en la que vamos a considerar tareas, puertas *and* (*parallel gateways*), *or* (*exclusive gateways*) y recursos.

Por ejemplo, el diagrama de la figura describe el proceso de negocio de una empresa de reparto a domicilio.

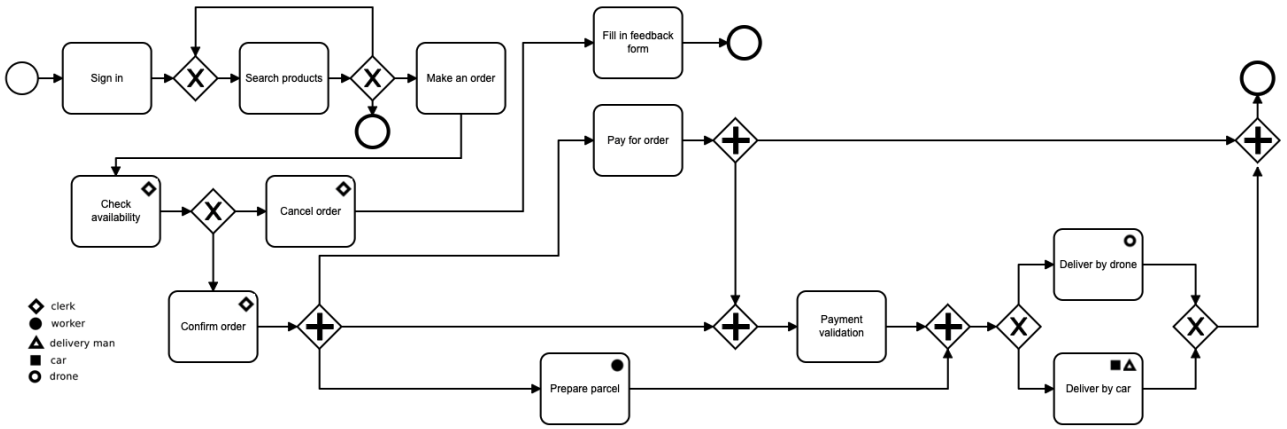

Figura 1: Reparto de paquetes

La ejecución de las tareas requiere unos recursos, que para simplificar supondremos especificados en unidades (p.ej., 3 drones o 3 personas). Y para representarlo gráficamente utilizaremos iconos como en la figura para indicar qué recursos, y en qué cantidad, son necesarios para la ejecución de una tarea. La ejecución de las tareas tiene una duración, y la comunicación entre tareas puede tener un retraso.

Para poder modelar el funcionamiento de un proceso de negocio como el de la figura, proporcionaremos representaciones de tareas, flujos, puertas y recursos que nos permitirán analizar su funcionamiento. La ejecución de un proceso la modelaremos como tokens que van moviéndose sobre el proceso, activando aquel flujo, puerta o tarea sobre el que se sitúan. Para hacerlo, utilizaremos términos de la forma *token(Id, T)*, donde *Id* es el identificador del elemento en el que está el token (flujo, puerta o tarea), y T es el tiempo necesario para que el token pueda ser utilizado. Por ejemplo, si al comenzar la ejecución del proceso anterior tenemos un token en el nodo inicial, una vez su tiempo se hace 0, el token pasará al flujo de salida de dicho nodo inicial. Una vez su temporizador se hace 0, se puede comenzar la ejecución de la tarea *Sign in*. Esta tarea también tendrá una duración, pero al acabar, se pasará el token al flujo de salida de dicha tarea. Cuando el token en el flujo de salida de la tarea se hace 0, se activa la puerta *or* (*exclusive merge gateway*). Observa que esta puerta indica que si tenemos un token (con tiempo 0) en cualquiera de los flujos de entrada de la puerta, el token pasa a su flujo de salida, con el temporizador inicializado al valor de retraso de dicho flujo. Cada una de estas acciones se modelará como una regla de reescritura.

Una puerta con un flujo de entrada y varios de salida se denomina un *split*. En una puerta *split or* (denotado con un 'por' gráficamente) la activación de la puerta se consigue cuando tenemos un token con temporizador 0 en su flujo de entrada, lo que produce que se pase el token a uno de los flujos de salida. En un *split and* (denotado con un 'más' gráficamente), se crea un token en cada uno de los flujos de salida. Las puertas con varios flujos de entrada y un único de salida se denominan *merges*. Un *merge or* funciona de la misma forma: la puerta se actica si hay un token en uno de los flujos de entrada. En un *merge and*, un token en cada uno de los flujos de entrada activa la puerta.

Una tarea puede necesitar algún tipo de recursos para activarse. Más concretamente, una tarea

puede necesitar una o varias instancias de un mismo recurso, o varias instancias de distintos recursos. Gráficamente, los recursos están indicados como iconos del tipo correspondiente en cada tarea.

Identificadores

Los identificadores de flujos y nodos serán de la forma *id("...")*.

```
op id : String -> Id [ctor] .
```

Nodos, flujos y puertas

Tendremos dos tipos de puertas, *exclusive* y *parallel*.

```
sort Gateway .
op exclusive : -> Gateway [ctor] .
op parallel : -> Gateway [ctor] .
```

Y seis tipos de nodos, que representaremos con los siguientes constructores:

- Un nodo inicial se representa como un término `start(Id1, Id2)`, con `Id1` el identificador del nodo e `Id2` el identificador de su flujo de salida.

  ```
  op start : Id Id -> Node [ctor] .
  ```

- Un nodo final se representa como un término `end(Id1, Id2)`, con `Id1` el identificador del nodo e `Id2` el identificador de su flujo de entrada.

  ```
  op end : Id Id -> Node [ctor] .
  ```

- Una tarea se representará como `task(NId, TaskName, FId1, FId2, T, RIds)`, donde `NId` es el identificador de la tarea, `TaskName` es la descripción de la misma (el texto que aparece en cada tarea en la figura), `FId1` y `FId2` son los flujos de entrada y salida de la tarea, `T` es la duración de la tarea y `RIds` es el conjunto de los identificadores de recursos que necesesita la tarea.

  ```
  op task : Id String Id Id Time Set{Id} -> Node [ctor] .
  ```

- Un *split* de tipo `G` `split(NId, G, FId, FIds)` tiene identificador `NId`, flujo de entrada `FId` y flujos de salida `FIds`.

  ```
  op split : Id Gateway Id Set{Id} -> Node [ctor] .
  ```

- Un *merge* de tipo `G` `split(NId, G, FIds, FId)` tiene identificador `NId`, flujos de entrada `FIds` y flujo de salida `FId`.

  ```
  op merge : Id Gateway Set{Id} Id -> Node [ctor] .
  ```

Recursos

Supondremos que un proceso tiene disponibles una cantidad limitada de recursos de cada tipo. En cada momento de la ejecución, algunos de ellos estarán en uso. Cada tipo de recurso se representará como un término `resource(Id, N, M)`, donde `Id` es el identificador del recurso, `N` el número total de instancas de ese recurso y `M` las unidades disponibles en un momento determinado.

```
sort Resource .
op resource : Id Nat Nat -> Resource [ctor] .
```

Flujos

Los flujos tienen un identificador y un retraso.

```
op flow : Id Time -> Flow [ctor] .
```

Tokens

Un token tendrá la forma *token(Id, T)*, donde *Id* es el identificador del elemento en el que está el token (flujo o tarea), y T es el tiempo necesario para que el token pueda ser utilizado.

```
op token : Id Time -> Token [ctor] .
```

La clase `Process`

Una instancia de la clase `Process` contendrá toda la información necesaria para la representación, ejecución y análisis de un proceso de negocio: un conjunto de nodos, un conjunto de flujos, los tokens que representan el estado de ejecución del proceso, el tiempo global del sistema y los recursos del sistema.

```
class Process | nodes: Set{Node},
                flows: Set{Flow},
                tokens: Set{Token},
                gtime: Time,
                resources: Set{Resource} .
```

Un token activará una acción sólo si su temporizador está a cero. Por ejemplo, los flujos tienen un retraso y las tareas tienen una duración. Cuando un token se pone en un flujo este se pone con su temporizador inicializado al valor de retraso del flujo. Mientras el temporizador no sea cero, este no podrá ser utilizado, modelando así la duración asociada a dicho flujo. Algo parecido ocurrirá con una tarea, aunque en este caso, el comienzo de la ejecución de una tarea requerirá de un token con temporizador a cero en su flujo de entrada y disponer de todos los recursos necesarios. Una vez se den las condiciones, el token será movido a la tarea, y cuando su temporizador sea cero, la ejecución de la misma terminará y el token podrá ser movido al flujo de salida de la misma. Lo mismo sucederá con las puertas, aunque dependiendo del tipo de puerta el comportamiento será uno u otro. Por ejemplo, cuando hay un token con temporizador a cero en el flujo de entrada de una *split parallel*, su activación producirá un token en cada uno de los flujos de salida, con los tiempos de retraso de cada una de dichos flujos. La activación de un *merge paralelo* requiere un token con temporizador cero en cada uno de sus flujos de entrada.

La gestión del tiempo se gestionará con una regla `tick`, y funciones `mte` y `delta` correspondientes. Obsérvese que sólo los tokens tienen temporizadores. Los conjuntos de nodos y flujos de los atributos `nodes` y `flows` no se verán alterados a lo largo de la ejecución de un proceso.

El atributo `resources` contendrá el estado de los recursos.

Se pide:

1. Especificad las reglas de reescritura definiendo el comportamiento de un proceso como el descrito. Necesitaréis una regla por cada una de las acciones que pueden ocurrir: comienza una tarea, termina una tarea, se activa un *exclusive merge*, ...

   Tened en cuenta que:

   - Un token activa el flujo, tarea o puerta en que se encuentra cuando su temporizador se hace 0.
   - Para que se dispare una puerta *and/parallel* se necesitan tokens en todos sus flujos de entrada, y todos con temporizador 0.
   - La ejecución de una tarea requiere un token con temporizador 0 en su flujo de entrada, y todos los recursos especificados en su descripción.
   - Cuando un token alcanza un nodo final podemos eliminarlo.
   - Para usar el tipo `Time` podéis importar el módulo `POSRAT-TIME-DOMAIN-WITH-INF` (tiempo denso con infinito) que tenéis en el fichero `time.maude`.

2. Utiliza la especificación del proceso del fichero `ex-process-3.maude`, donde se proporciona una constante `PROCESS` que especifica un proceso de negocio según las especificaciones de la tarea de la práctica, con un token en su nodo inicial, para:

   - Utiliza el comando `search` para buscar estados del proceso a lo largo de su ejecución limitada a 100 unidades de tiempo en que no haya ningún token en el conjunto del atributo `tokens`. Explica el resultado.
   - Utiliza el comando `search` para verificar si hay situaciones de bloqueo para ejecuciones del proceso antes del transcurso de 100 unidades de tiempo. Explica el resultado.