Formalización de modelos de simulación

PID_00249440

Pau Fonseca Casas

Tiempo mínimo de dedicación recomendado: 4 horas

Universitat Oberta de Catalunya

Ninguna parte de esta publicación, incluido el diseño general y la cubierta, puede ser copiada, reproducida, almacenada o transmitida de ninguna forma, ni por ningún medio, sea éste eléctrico, químico, mecánico, óptico, grabación, fotocopia, o cualquier otro, sin la previa autorización escrita de los titulares del copyright.

Índice

1. Introducción 5

2. Señales en SDL 10
2.1. *Delaying channels* 11
2.2. *Non-delaying channels* 11
2.3. *Signal delay* 12

3. Definiendo el comportamiento, el diagrama de proceso 13
3.1. Procedimientos 15

4. Ejemplo, el contenedor químico 17

5. Ejemplo, el agente reflexivo simple 23
5.1. Especificación usando SDL 24
5.2. Codificación 29

6. Otros lenguajes de formalización de modelos de simulación 30
6.1. DEVS 30
6.1.1. Ejemplo: el procesador 33
6.1.2. Ejemplo, la cola FIFO 34
6.1.3. Modularidad en DEVS 35
6.2. Redes de Petri 36
6.2.1. Modularidad en las RdP 40

Bibliografía 43

1. Introducción

Specification and Description Language, en adelante SDL, es un lenguaje estándar gráfico, completo y no ambiguo estandarizado por la ITU-T, una de las organizaciones de estandarización más antiguas y reconocidas que existen actualmente. De hecho, las tres organizaciones de estandarización más reconocidas son:

- International Organization for Standardization (ISO), fundada en 1947
- International Electrotechnical Commission (IEC), fundada en 1906
- International Telecommunication Union (ITU), fundada en 1865

Todas están ubicadas en Ginebra, Suiza, y configuran conjuntamente la World Standards Cooperation (WSC) Alliance.

El propósito inicial de SDL fue la de ser un lenguaje no ambiguo de especificación y descripción para la estructura, el comportamiento y los datos de los sistemas de telecomunicaciones.

El lenguaje se define a partir de recomendaciones que propone la ITU-T. Las diferentes recomendaciones que existen son:

- Z.100 (11/99) Specification and description language (SDL) including various annexes and language (SDL) including various annexes and language (SDL) including various annexes and language (SDL) including various annexes and appendices
- Z.105 (11/99) SDL combined with ASN.
- Z.107 (11/99) SDL with embedded ASN.1
- Z.109 (11/99) SDL combined with UML

Usualmente, SDL se ha usado para la especificación de sistemas reactivos, en el ámbito del área aeroespacial, el control automático, sistemas médicos, electrónica, telecomunicaciones, simulación y más recientemente para generar directamente código del producto, a través de sistemas como *PragmaDev Studio* entre otros.

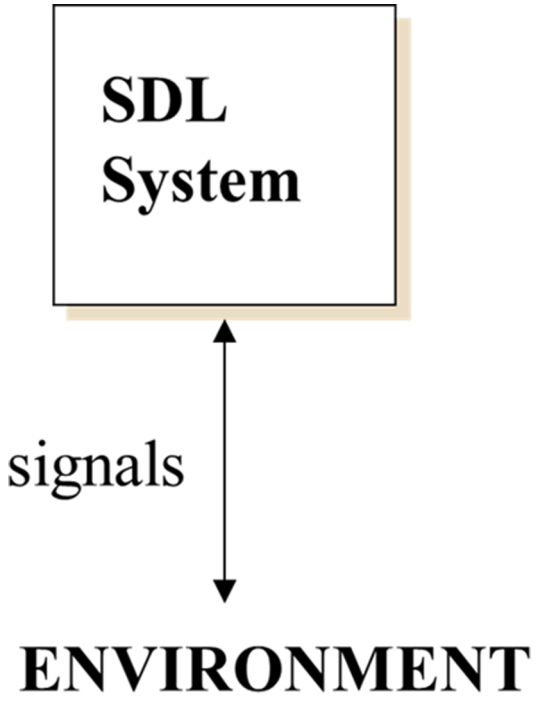
El lenguaje ha tenido una evolución larga que le ha permitido evolucionar acorde a las necesidades de la industria. Las diferentes ediciones de la especificación son:

- 1976 Orange Book SDL, Basic graphical language
- 1980 Yellow Book SDL, Process semantics defined
- 1984 Red Book SDL, Structure, data added. Definition more rigorous.Start of tools. User guide
- 1988 Blue Book SDL (SDL-88). Effective tools. Syntax well defined - formal definition. Language much as 1984
- 1992 White Book SDL-92. Object SDL. Types for blocks, processes, services with inheritance and parameterisation. Methodology guidelines
- 1995 SDL with ASN.1 (Z.105)
- 1996 Addendum 1 to SDL-92
- Language stable. Some relaxation of rules. SDL+ Methodology. Tools offer SDL-92 features
- 1999 SDL-2000. Object modeling support.Improved implementation support. Data model revised
- 2012 SDL-2010

La especificación actual se puede consultar en ITU-T (2011), mientras que algunos libros interesantes que permiten entender cómo funciona el lenguaje son Doldi (2001) y Fonseca Casas (2014a).

SDL se compone de dos partes, una parte estática y una parte dinámica; la parte estática describe la estructura del modelo a representar, las entidades y subentidades y cómo estas se conectan, así como las señales que se usan para establecer esta comunicación. La parte dinámica describe el comportamiento, las transiciones y las comunicaciones efectivas entre los diferentes componentes, así como los procesos y su vida y muerte.

Figura 1. Sistema y entorno en un modelo SDL.



La especificación SDL define cómo el sistema reacciona a los acontecimientos que tienen lugar en el entorno, que se comunican por señales enviadas al sistema. La única forma de comunicación de un sistema SDL con el entorno es a través de señales.

El lenguaje presenta una estructura modular, tal como se puede ver en la figura 2.

SDL tiene dos sistemas de representación, SDL-GR, representación gráfica, usada como la representación usual y concebida para ser entendida fácilmente por los humanos. SDL-PR, representación textual, pensada para los ordenadores y para simplificar su intercambio de información. Las dos representaciones son equivalentes y en este texto usaremos únicamente SDL-GR.

Los diferentes niveles que podemos encontrar en una especificación SDL son:

- Diagrama de sistema; único, representa la parte más exterior del modelo y muestra la relación a través de señales (SIGNALS) que existe entre el modelo y el entorno (ENVIRONMENT).

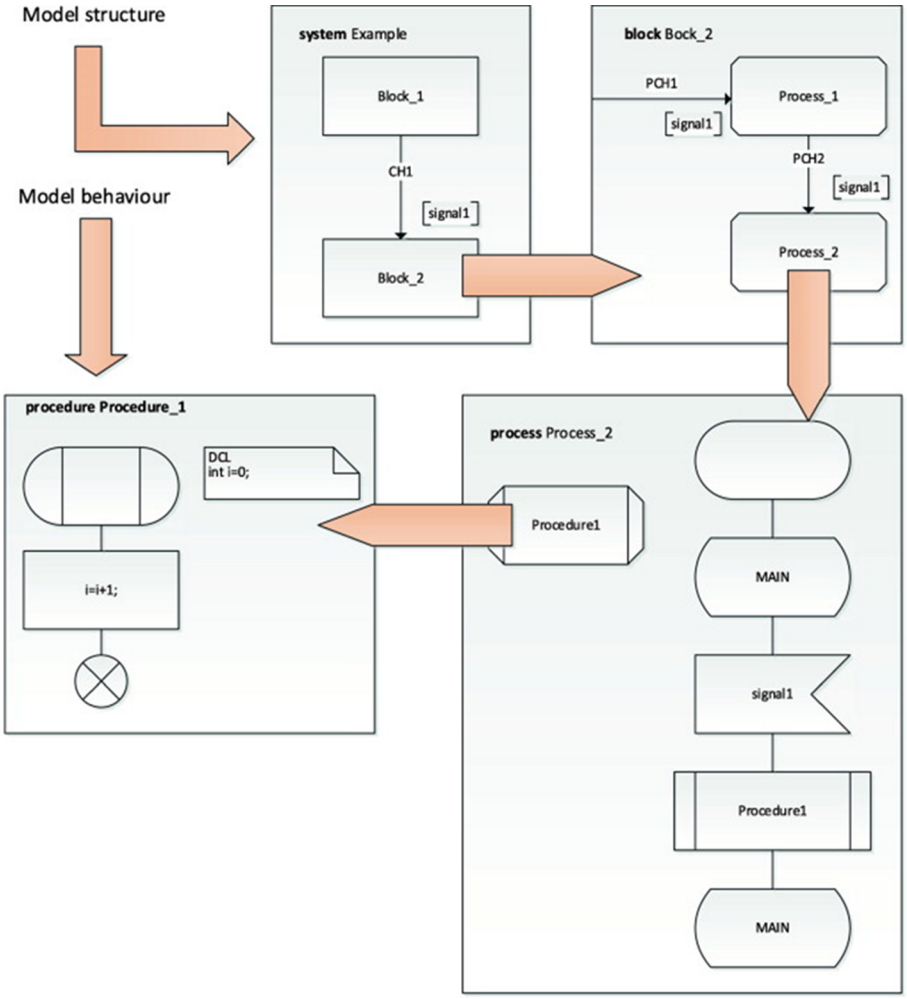
- Diagrama de bloques. En el diagrama de sistema se pueden encontrar uno o más bloques que permiten establecer la descomposición jerárquica del modelo. Esta descomposición permite estructurar los diferentes elementos de forma acorde al objetivo del modelo. Un bloque se puede descomponer en nuevos bloques; por tanto, en una especificación SDL es usual en-

contrar una descomposición jerárquica compuesta por varios diagramas de bloques.

- Diagrama de procesos. Una vez se ha acabado con la descomposición jerárquica de los elementos principales del modelo, a través de los diagramas de bloques, se ha de pasar a definir el comportamiento de los mismos. Esto se logra a partir de los diagramas de procesos. Estos diagramas se pueden descomponer en nuevos diagramas de proceso, pero no en diagramas de bloques. Es decir, una vez empezamos a descomponer el modelo en diagramas de proceso, no podemos incorporar en su interior nuevos diagramas de bloques.

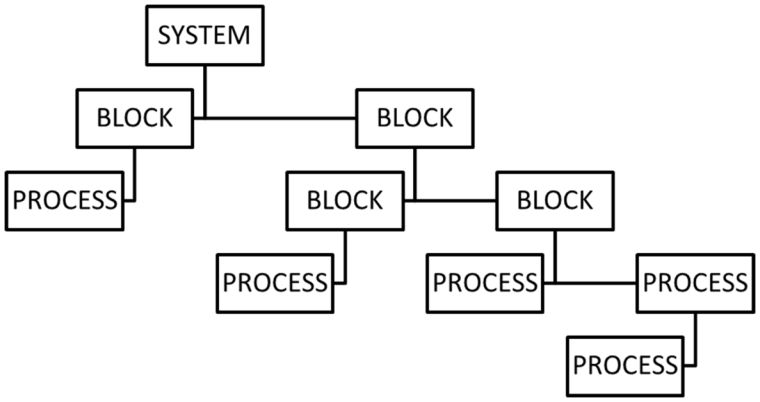
- Procedimientos, que permiten estructurar (agrupar) el código y el comportamiento que encontramos en los diagramas de proceso.

Figura 2. Estructura modular del lenguaje SDL.



Esta descomposición jerárquica se puede ver representada en la figura 3.

Figura 3. Descomposición jerárquica de los elementos SDL.

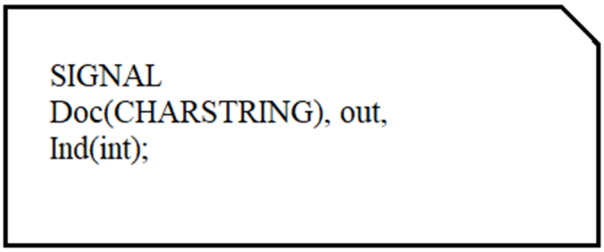


El SYSTEM, así como los diagramas de bloques y los de procesos, se identifican con agentes; los procedimientos que pueden estar contenidos en los PROCESS no lo son.

2. Señales en SDL

La definición de las señales se hace en los TEXT, que podemos encontrar en cualquiera de los diagramas (ver figura 4).

Figura 4. Definición de las SIGNALS en los bloques TEXT.

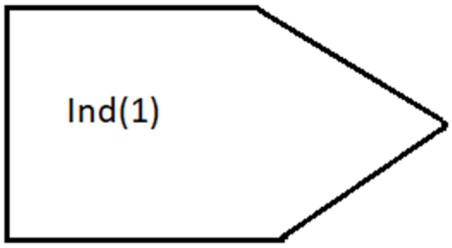


En este ejemplo se pueden ver 3 señales, denominadas Doc, out y Ind. Tanto Doc como Ind tienen un parámetro, es decir, transportan información entre dos agentes, mientras que out no transporta información (aparte de la inherente información contenida en la transmisión de la señal).

La definición de las señales se hace allí donde se necesitan, es decir, en el nivel del diagrama SDL donde aparecen por primera vez. En los siguientes niveles de la especificación no hará falta definir las señales de nuevo.

El envío de las señales tiene lugar en los PROCESS, dado que son elementos intrínsecos del comportamiento del modelo. En la figura 5 se puede ver el símbolo que usaremos para enviar una señal en el punto del proceso que lo requiera.

Figura 5. Envío de una señal.



El envío de la señal seguirá los CHANNELS, canales de comunicación que tengamos definidos en los diagramas. Si no existen canales de comunicación, la señal no viajará a otro agente y por tanto no tendrá efecto. Es por ello que uno de los objetivos principales que se persigue en la definición de la estructura del modelo, a través de su descomposición jerárquica, es la definición de los canales que han de permitir establecer una correcta comunicación entre los

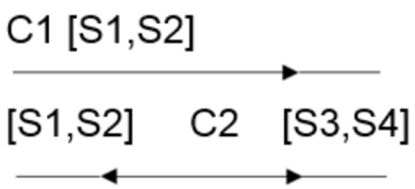
diferentes elementos del modelo. En la figura 6 se puede ver un canal con el nombre de «channel» que permite transmitir las siguientes señales, T_MOD (hacia la derecha) y T_OK, T_HI y T_CR (hacia la izquierda).

A continuación, vamos a detallar un poco más cómo podremos definir los canales de comunicación entre los diferentes agentes y sus características.

2.1. *Delaying channels*

Los primeros canales que hay que considerar son los *delaying channels* o canales con demora. Son canales que requieren de un tiempo para poder transmitir la señal. El aspecto que presentan estos canales se puede observar en la figura 6. Como puede observarse, podemos tener canales unidireccionales y bidireccionales, y podemos especificar las señales que pueden viajar por el canal (especificando el sentido de la transmisión de las señales).

Figura 6. Canales con demora.

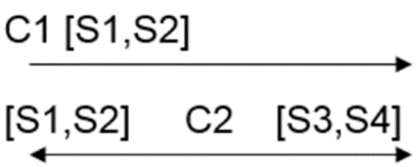


Los *delaying channels* no permiten especificar la duración de la demora y por tanto no se usarán usualmente para modelar un sistema en el que queremos controlar las demoras, por ejemplo, usando distribuciones de probabilidad. Para modelar estos casos, usaremos los *non-delaying channels* en combinación con el *signal delay*, tal como se explicará a continuación.

2.2. *Non-delaying channels*

Los canales sin demora (ver figura 7) transmiten las señales de forma instantánea.

Figura 7. Canales sin demora.



Su definición gráfica es muy similar a los *delaying channels*, con la particularidad de que las puntas de la flecha ahora están situadas al final de los canales. Dado que usualmente en los modelos tendremos que establecer una demora en la transmisión de las señales, estos canales tendremos que combinarlos con el *signal delay*, que permitirá definir el tiempo de tránsito necesario para que la

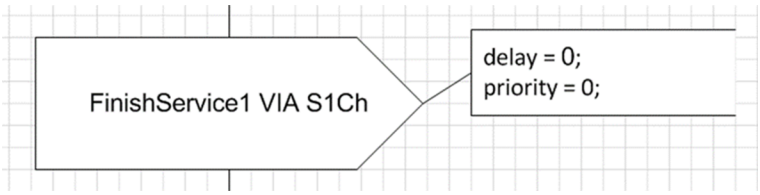
señal llegue a su destino. Este tiempo es independiente del número de canales que tendrá que atravesar y dependerá únicamente de la instrucción de envío de la señal (OUTPUT).

2.3. *Signal delay*

A partir de la especificación SDL-2010, se permite que las señales, en su instrucción OUTPUT, definida en los procesos, indiquen el tiempo necesario (y la prioridad) para llegar a su destino.

Esta definición es muy simple, tal como se muestra en la figura 8. Como se puede ver, simplemente hay que especificar, en el campo reservado *delay*, el tiempo necesario para transmitir la señal. En este ejemplo el tiempo es 0, por tanto, la señal se transmite de forma instantánea. En vez de 0 podríamos haber puesto cualquier función que devolviera un valor real positivo para indicar el tiempo necesario para transmitir la señal.

Figura 8. Definición de una demora en una señal usando el *signal delay*. En este caso, dado que la demora es 0 no tiene efecto.

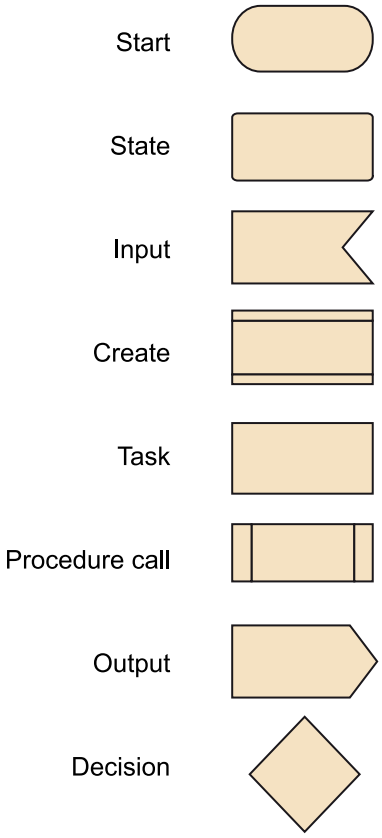


3. Definiendo el comportamiento, el diagrama de proceso

El diagrama de proceso (*process diagram*) es la herramienta fundamental para definir el comportamiento del modelo. Una vez establecida su estructura y su descomposición jerárquica, así como la comunicación entre cada una de sus partes a través de los canales, el diagrama de proceso permite definir el comportamiento del modelo.

El diagrama de proceso define el comportamiento de un agente a partir de la recepción de una señal específica. Un diagrama de proceso usa diferentes elementos gráficos (ver figura 9) para detallar este comportamiento a partir de un flujo de datos (una máquina de estados extendida).

Figura 9. Elementos gráficos del diagrama de procesos.



Estos elementos gráficos son:

Start. Este elemento define las condiciones iniciales del diagrama de proceso. Es única para cada diagrama y representa el punto de partida del diagrama. A partir de él se empieza a ejecutar el modelo.

State. El elemento State contiene el nombre del estado. Este elemento define los diferentes estados en los que se puede encontrar el diagrama de proceso. Un diagrama de proceso siempre tendrá que estar en un estado o en otro, con la excepción del momento inicial, cuando el proceso se instancia. En ese instante inicial pasará del Start al estado inicial ejecutando las acciones necesarias para inicializar el proceso.

Input. El elemento Input describe el tipo de señal (evento) que puede ser recibida por el proceso para un estado determinado. Este elemento se encuentra siempre a continuación de un State, es decir, sirve para determinar cómo pasaremos de un estado a otro.

Create. Este elemento permite la creación de un agente (elemento BLOCK o PROCESS).

Task. Este elemento permite interpretar código en su interior. En la versión de SDL-RT (PragmaDev SARL, 2006), se puede usar código C, de la misma forma que a partir de SDL-2010.

Procedure call. Este elemento permite llamar a un procedimiento. El PROCEDURE tendrá que ser definido en el último nivel de la especificación SDL. Sirve para encapsular piezas de código y rehusarlas. Usualmente, también se usa para encapsular aquellos elementos del modelo que no requieren una especial atención en el proceso de validación (pero que hay que verificar).

Output. Este elemento lo hemos comentado brevemente en la sección 2.3; permite enviar una señal a otro PROCESS. Permite definir el nombre de la señal, así como los parámetros que tiene asociados. Si existe ambigüedad respecto a quién puede finalmente procesar la señal, se pueden usar elementos de redireccionamiento para asegurarnos de que la señal llega a su destino. Entre estos elementos podemos usar el PId (Process Identifier), que es un identificador no ambiguo que todo proceso tiene (la especificación solo asegura que todo PROCESS tiene un PId único, pero no indica cómo definir este PId).

Se puede especificar también el nombre del canal que se quiere usar para enviar la señal, usando la instrucción VIA. Por ejemplo, **Out VIA channel_1** enviaría la señal Out usando el canal identificado por channel_1.

Si existe más de una alternativa posible, se escoge una de forma arbitraria. El destino que se ha aplicado a una señal se guarda para un uso posterior. Cuatro expresiones que guardan el PId pueden usarse:

- **self**, el propio PId del proceso que está procesando la señal.

- **parent**, el PId del agente que ha creado el agente que está procesando la señal. (Null para agentes iniciales).

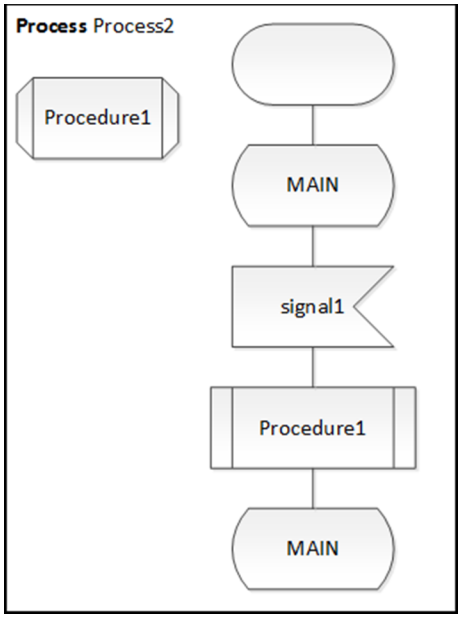
- **offspring**, el último agente creado por el agente que procesa la señal (con l elemento CREATE).

- **sender**, el PId del agente que ha enviado la señal. (Null antes de la recepción de ninguna señal).

Decision. Estos elementos permiten describir bifurcaciones en los procesos. Su comportamiento depende de cómo la pregunta es contestada.

La figura 10 muestra un proceso que contiene el elemento START, que define que el estado inicial para este proceso será MAIN. Se puede ver que en este estado únicamente se espera la recepción de la señal *signal1*, que desencadenará la ejecución del procedimiento *procedure1* y nos devolverá, en este caso, al mismo estado.

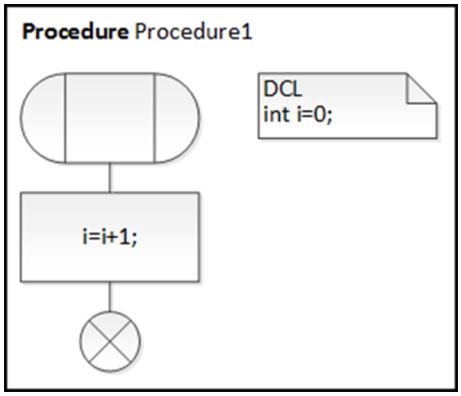
Figura 10. Diagrama de proceso para el estado MAIN.



3.1. Procedimientos

El último nivel de una especificación SDL se define a partir de los procedimientos (PROCEDURES). Como hemos comentado, se podrán usar dentro de los procesos a partir de los PROCEDURE CALLS. Estos diagramas son muy parecidos a los diagramas de proceso, con la excepción de que no necesitan definir estados. La figura 11 muestra un ejemplo de PROCEDURE. Hay que notar el símbolo inicial, que indica el inicio del procedimiento, y la cruz final, que marca el final del mismo. Usualmente, el paso de parámetros se hace por referencia.

Figura 11. Definición de un PROCEDURE.



4. Ejemplo, el contenedor químico

En este ejemplo vamos a considerar una fábrica de productos químicos en la que se produce un producto tóxico altamente inflamable. El objetivo principal del modelo es evaluar las diferentes alternativas de evacuación que existen para poder mejorarlas y determinar cuál aplicar en caso de emergencia.

En la sala específica que se va a modelar, existe un contenedor con el producto químico, en estado líquido, que puede cerrarse herméticamente en caso de emergencia. Si la temperatura del líquido contenido en el contenedor llega a un nivel «crítico», para evitar una fuga de gas tóxico, se procedería a provocar una explosión controlada del contenedor (esto se haría como último recuro y obviamente el modelo intenta encontrar los mecanismos para no llegar a esta situación).

Un sensor controla la temperatura y va indicando continuamente si esta es «normal», «alta» o «crítica».

Si la temperatura es alta, se procede a la evacuación y cierre de la sala; las puertas automáticas empiezan a cerrarse y el personal que eventualmente pudiera estar en la sala procede a salir de ella. En este punto se empieza a tirar un producto en el contenedor «Reactivo-B» con el objetivo de bajar la temperatura del líquido contenido en el contenedor.

Si la temperatura es crítica, independientemente de si el proceso de cierre de puertas y evacuación ha finalizado, se procede a realizar la explosión controlada.

Con estas premisas básicas, a continuación, se presenta la especificación SDL, que permite detallar el comportamiento de este sistema.

El primer diagrama que entonáramos en el diagrama de sistema, figura 12.

Figura 12. Diagrama de sistema representando el modelo del contenedor químico.

A partir de la definición del diagrama de sistema, se puede empezar a detallar cada uno de los elementos que nos hemos encontrado en él. En este caso, el BLOQUE *Container* y el BLOQUE *Sensor*. En la figura 13 se detalla en diagrama para el BLOQUE *Sensor* y en la figura 14 se detalla el diagrama para el BLOQUE *Container*.

Figura 13. Diagrama de BLOQUE para el sensor.

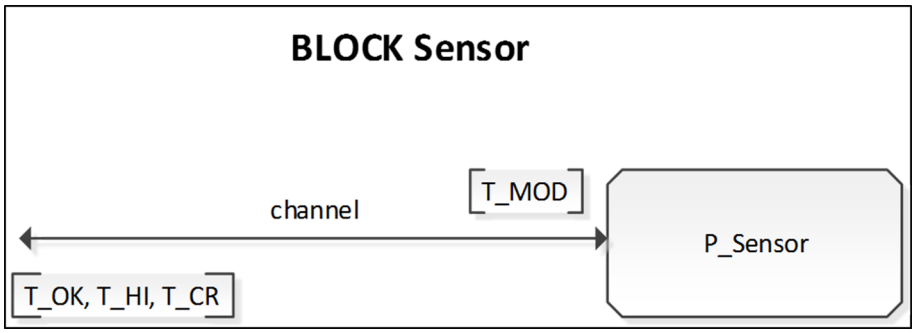
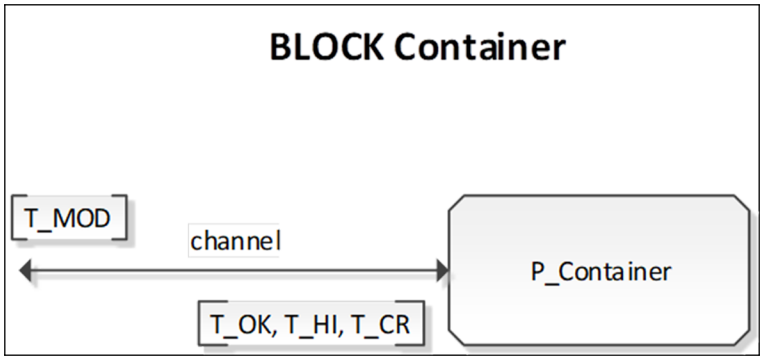


Figura 14. Diagrama de bloque para el contenedor.



En los dos diagramas de BLOQUE se puede apreciar que la descomposición presenta un único bloque de PROCESO; esto permite entender que el comportamiento de los dos elementos estará detallado en P_Sensor y P_Container respectivamente.

Para detallar este comportamiento, es interesante usar un diagrama de estados. El diagrama de estados no forma parte de la especificación SDL, pero es útil para entender, previo a la definición formal del modelo (y concretamente el diagrama de procesos), los estados que constituirán el subsistema, en este caso el sensor y el contenedor.

En la figura 15, se presenta el diagrama de estados para el sensor, y en la figura 16 el diagrama de estados para el contenedor.

Figura 15. Diagrama de estados para el sensor.

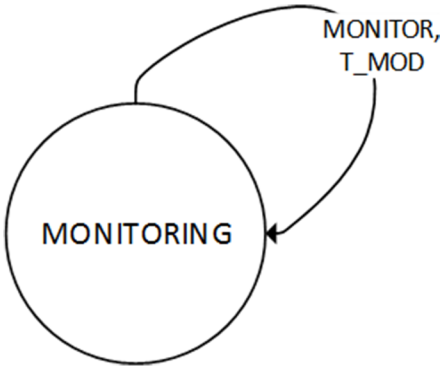
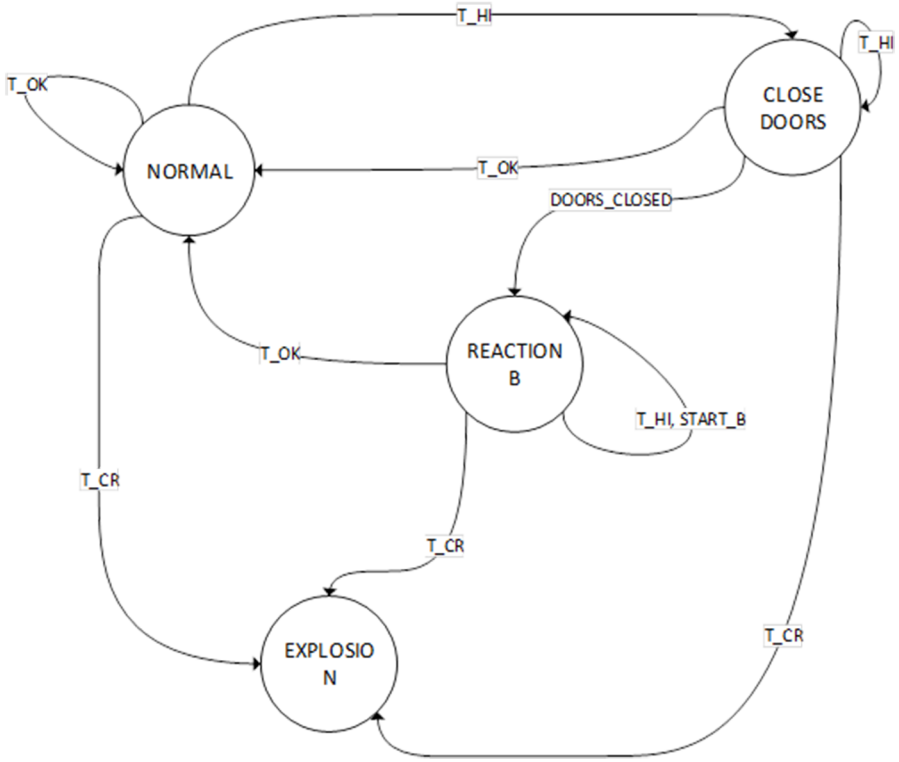


Figura 16. Diagrama de estados para el contenedor.



En este punto ya se puede especificar el comportamiento de cada uno de los diferentes elementos, el Sensor (P_Sensor) y el contenedor (P_Container). Esta definición está representada en la figura 17 y la figura 18 respectivamente.

Figura 17. PROCESO del sensor.

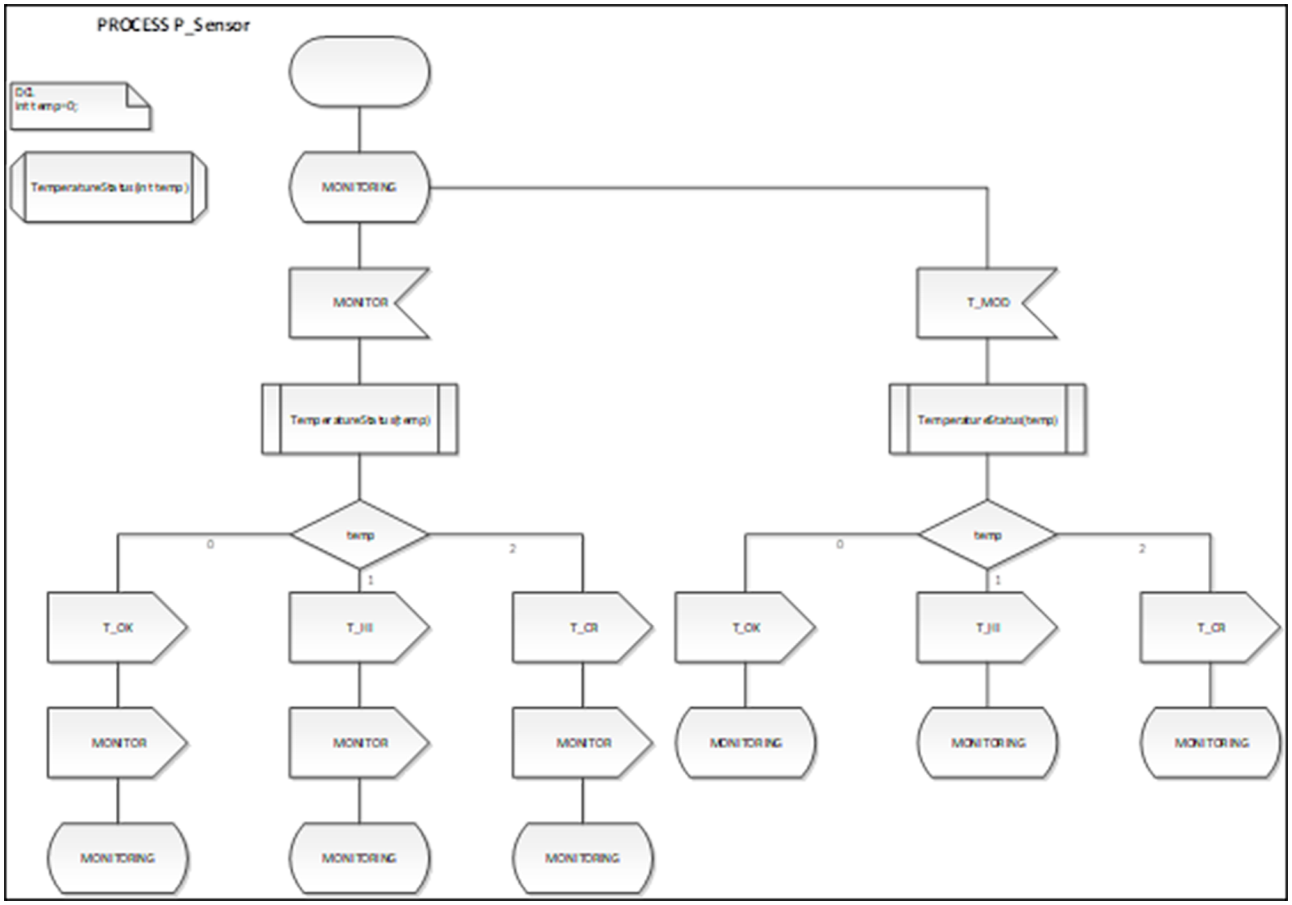
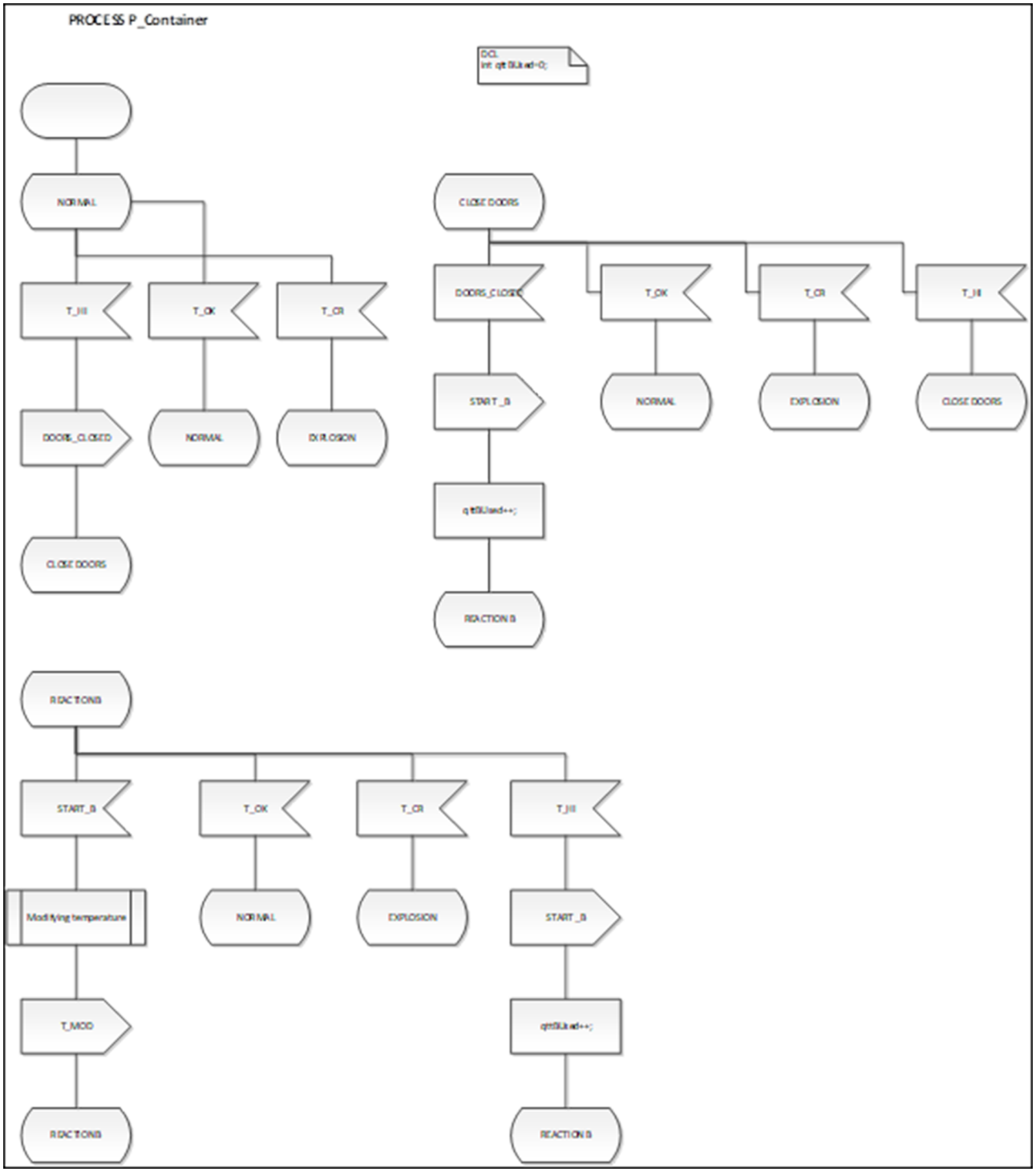
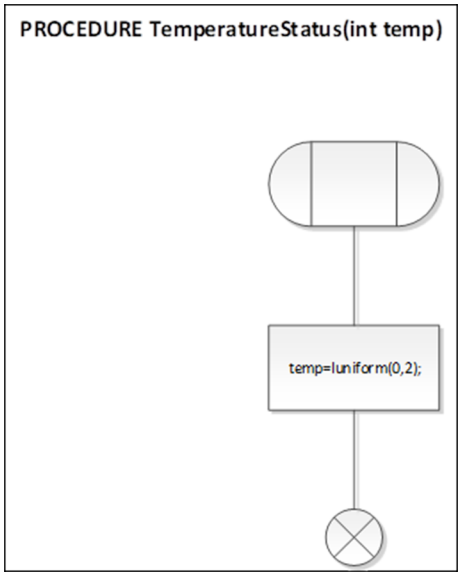


Figura 18. Diagrama de procesos para el contenedor.



Puede resultar interesante detallar gráficamente alguno de los PROCEDIMIENTOS asociados al comportamiento de los elementos. En este caso podemos llegar al último nivel de SDL. En la figura 19 se representa el PROCEDIMIENTO asociado al sensor, que permite determinar la temperatura. Es en este último nivel donde podríamos sustituir el elemento simulado con el elemento real para usar el modelo integrado con elementos reales del sistema.

Figura 19. Ejemplo de procedimiento para detectar la temperatura del sensor.

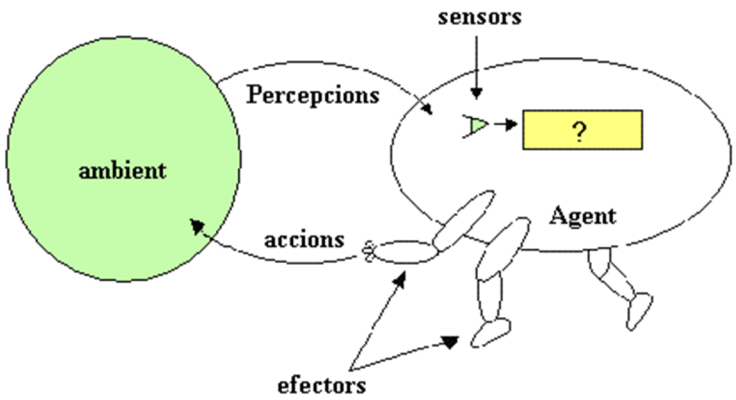


5. Ejemplo, el agente reflexivo simple

Este ejemplo está basado en Fonseca i Casas (2014b). En el ejemplo anterior, hemos visto una definición clásica de un sistema industrial usando SDL. En este ejemplo vamos a ver una definición de un agente reflexivo simple usando SDL. No es de extrañar que SDL sea un lenguaje excelente para representar agentes inteligentes dado que, en su concepción, el lenguaje pensó para representar sistemas reactivos distribuidos.

Lo primero que vamos a hacer es recordar que es un agente reflexivo simple. En la figura 21 se representa un esquema simplificado de un agente inteligente.

Figura 20. Esquema simplificado para un agente inteligente.

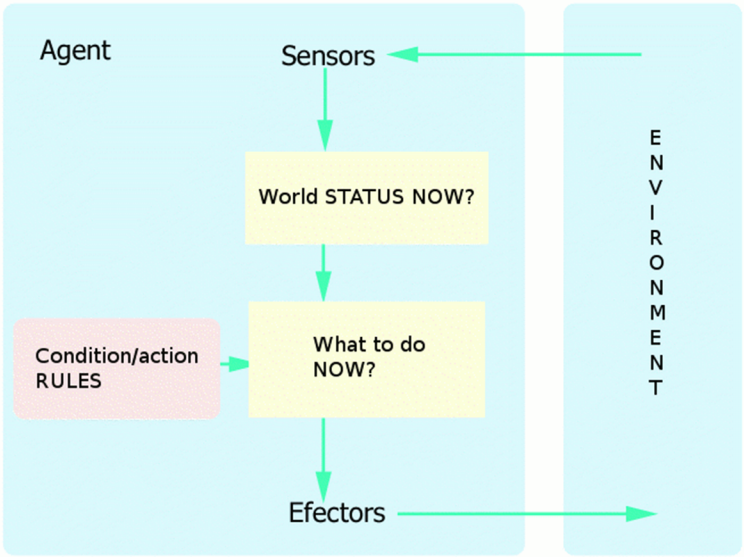


Un agente inteligente es un programa de ordenador que tiene, como principal particularidad, una capacidad limitada de acceso a la información global del sistema. Este esquema de funcionamiento es especialmente útil cuando con él se quiere representar sistemas que efectivamente tienen esta limitación.

Para establecer la comunicación con el entorno, todo agente inteligente cuenta con sensores, que le permiten obtener información, y con efectores, que les permiten realizar acciones sobre el entorno, sobre otros agentes o sobre el mismo agente.

A partir de la información que el agente recibe, en función del tipo de agente, este realizará un conjunto de acciones. El agente que vamos a representar formalmente es el agente reflexivo simple, un agente que por simplicidad permite entender perfectamente el funcionamiento de otros agentes con comportamientos más avanzados. En la figura 21 se puede ver el esquema de funcionamiento de un agente reflexivo simple.

Figura 21. Agente reflexivo simple.



El comportamiento básico de un agente inteligente reflexivo simple es el siguiente:

1) El agente espera a recibir alguna información del entorno. Cuando recibe alguna información, el agente la procesa y, en función de ella, modifica su conocimiento del entorno, que como hemos comentado es una información parcial.

2) El agente, a partir del conocimiento parcial que tiene, puede iniciar una o varias acciones que pueden modificar su entorno.

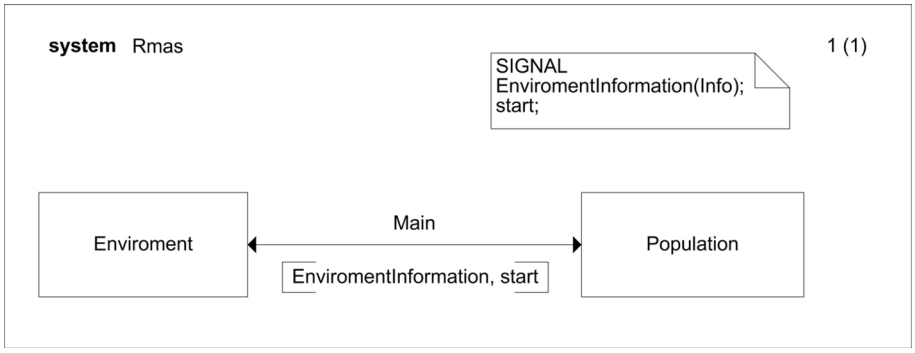
3) Una vez efectuadas las acciones, el agente vuelve a su estado de espera.

Con esta información podemos ya especificar el comportamiento de un agente reflexivo simple usando un lenguaje de especificación formal como SDL.

5.1. Especificación usando SDL

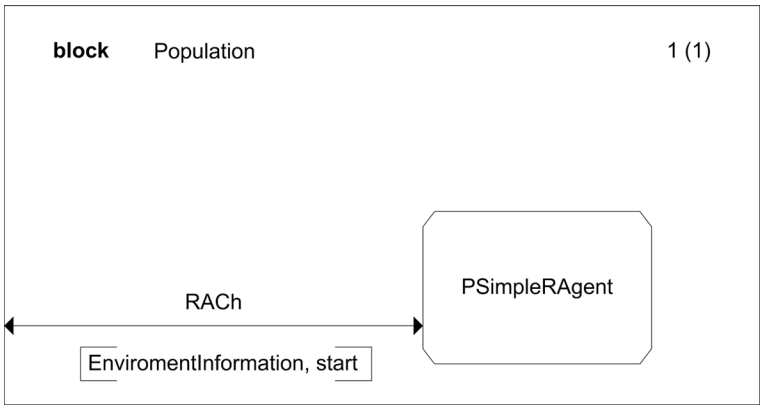
El primer diagrama a representar es aquel que nos ha de permitir explicar cómo el modelo se comunica con el exterior. En este caso, el diagrama de sistema (ver figura 22) nos permite especificar los mecanismos de comunicación del agente con el exterior. En este caso el agente, representado por el bloque Population, solamente establece comunicación con otro bloque que representa en entorno, bloque Environment, a través de un único canal denominado Main. Podemos ver también que las únicas señales que los dos AGENTES SDL intercambian son EnviromentInformation (con el parámetro Info) y start.

Figura 22. Diagrama de sistema para el modelo de agente inteligente reflexivo simple.



En este ejemplo, la descomposición jerárquica de los AGENTES SDL es simplemente un proceso que define su comportamiento. En la figura 23 se puede apreciar la descomposición jerárquica del AGENTE Population. El bloque se estructura simplemente en un único proceso denominado PSimpleRAgent, que definirá el comportamiento del agente.

Figura 23. Especificación del bloque Population que define la estructura del agente reflexivo simple.



Para el AGENTE Environment se podría especificar también su estructura (y posteriormente su comportamiento). Se puede consultar el artículo de referencia de este ejemplo para poder ver la especificación completa (Fonseca i Casas, 2014b).

A continuación, podríamos descomponer el proceso *PSimpleRAgent* que definirá el comportamiento del agente en otros subprocesos. En este caso no se considera necesario definir subprocesos y por tanto se procede a la especificación del comportamiento del mismo, tal como se puede ver en las siguientes figuras (ver figuras 24, 25 y 26).

Figura 24. *PSimpleRAgent* estado Inactive (inicial).

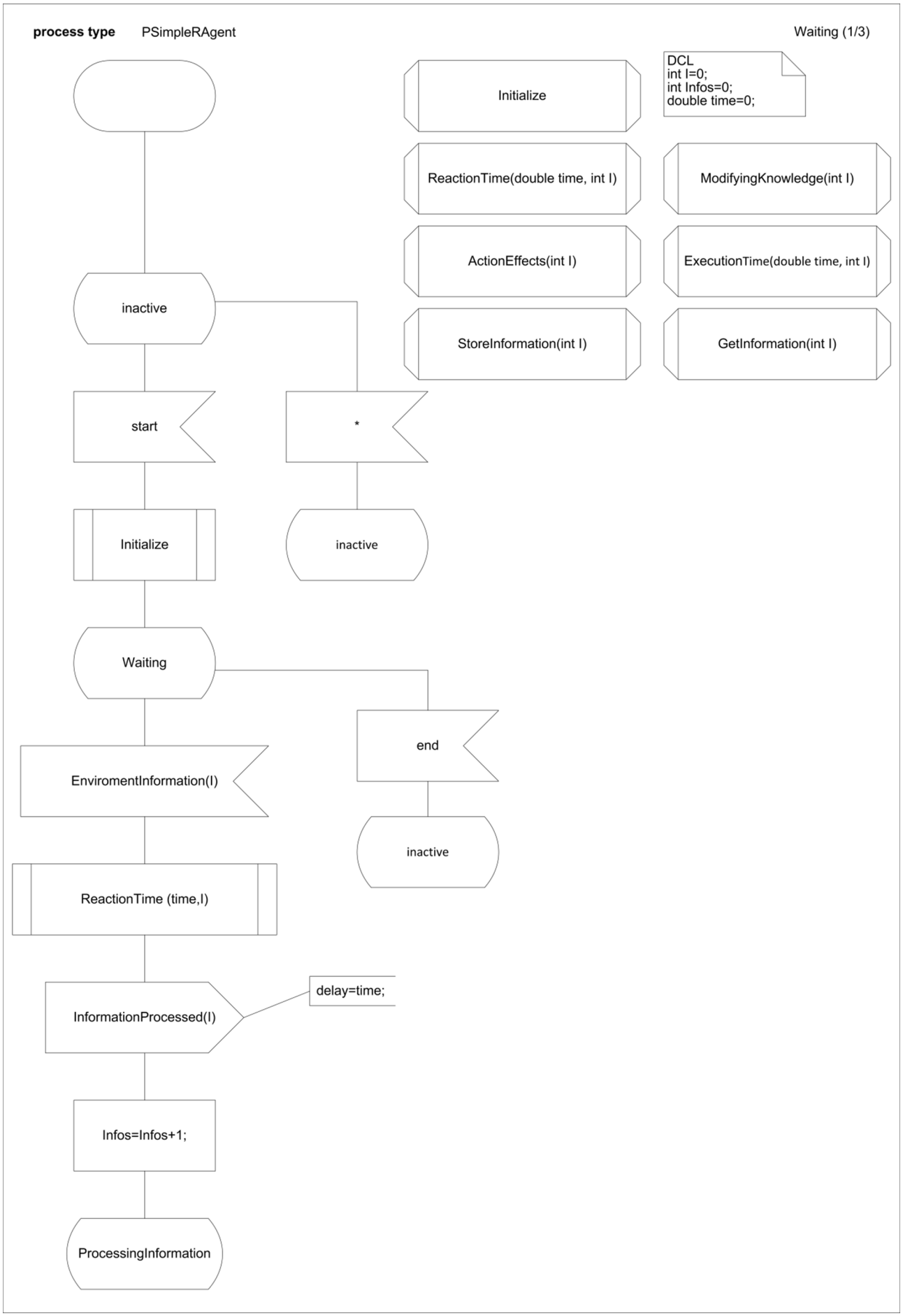


Figura 25. *PSimpleRAgent* en el estado ProcessingInformation.

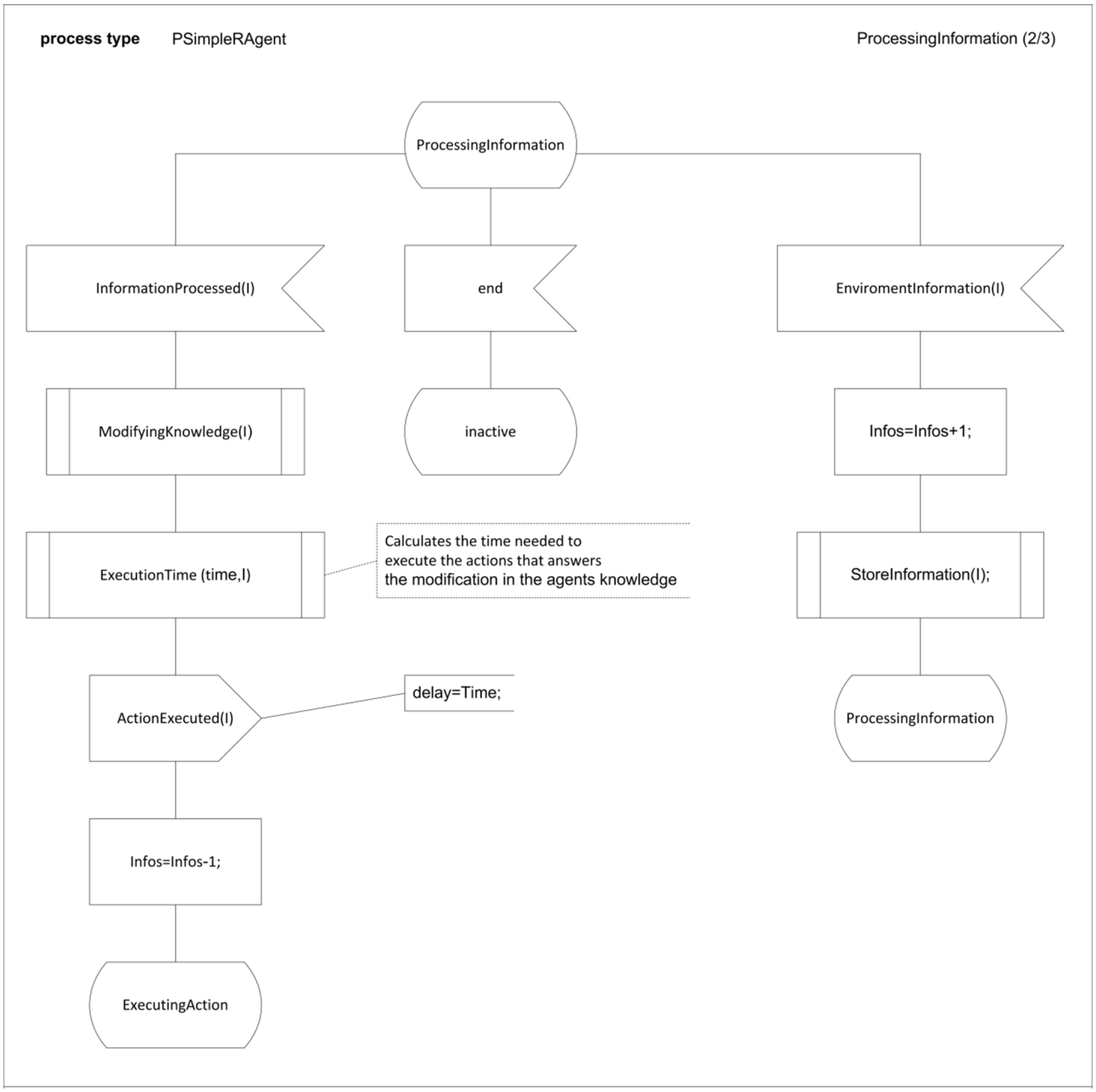
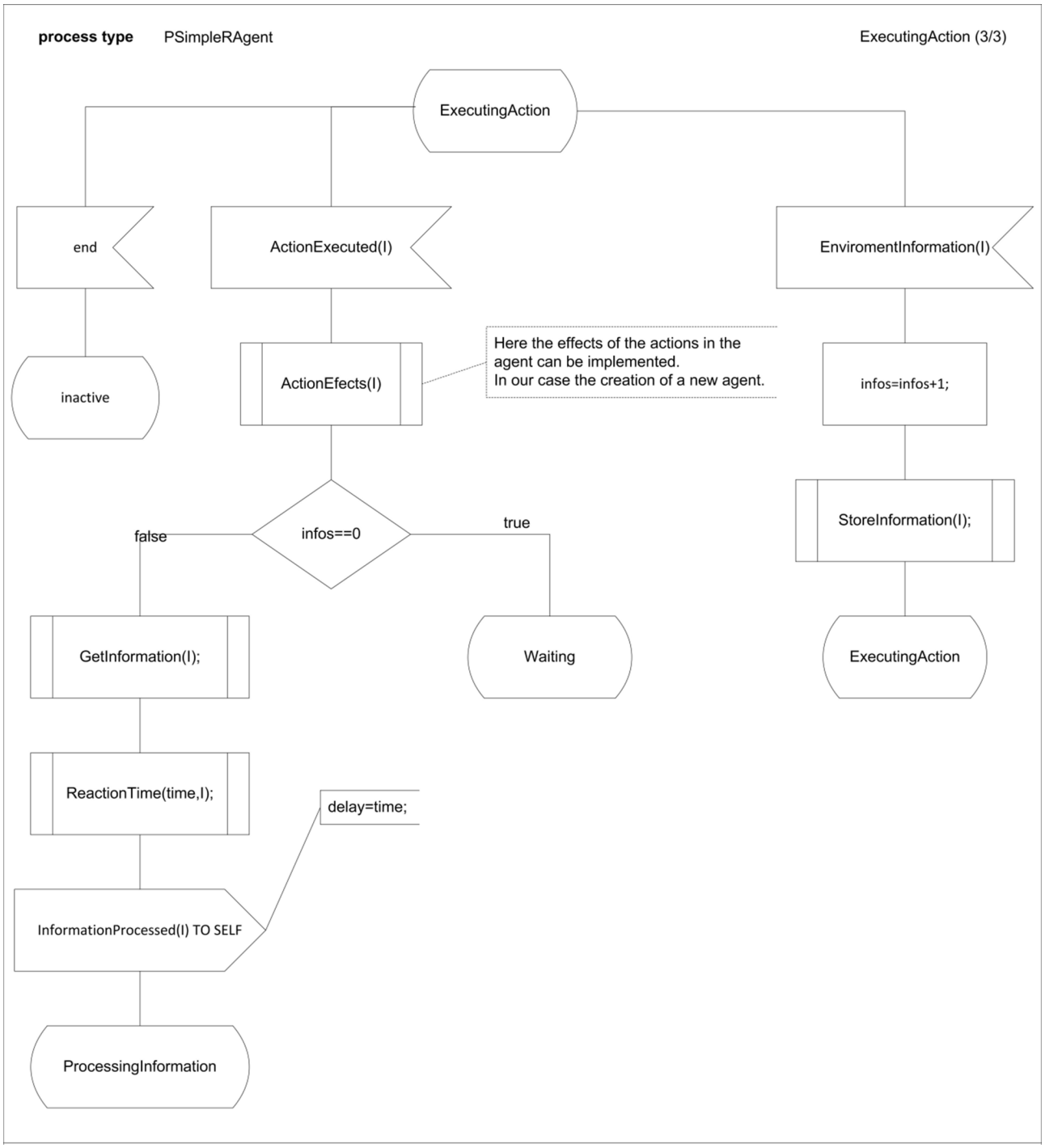


Figura 26. *PSimpleRAgent* en el estado ExecutingAction.



El comportamiento del agente queda completamente definido a partir de estos diagramas, lo que permite, a partir de su especificación, generar código de forma automática. En la siguiente sección haremos una breve descripción al respecto.

5.2. Codificación

A partir de una especificación usando un lenguaje completo y no ambiguo como SDL, podemos establecer mecanismos para generar, automática o semiautomáticamente, el código necesario para ejecutar el modelo.

Un ejemplo de generación automática de código, y dado que en este caso usamos SDL, es usar *PragmaDev Studio* o SDLPS (Fonseca i Casas, 2010), entre otras posibles soluciones. En Fonseca i Casas (2014b) se propone usar SDLPS como mecanismo de generación de código automático y NetLogo (Wilensky, 2016) como ejemplo usando un sistema específico para la implementación de sistemas multiagente. Se dejará como ejercicio la implementación de un agente inteligente completo usando alguna de las herramientas de tipo general que existen.

6. Otros lenguajes de formalización de modelos de simulación

SDL es un excelente lenguaje para especificar un modelo de simulación; no obstante, existen otras alternativas igualmente válidas para este mismo fin. Un libro a tener presente para completar esta sección es Fonseca i Casas (2014a), donde se recogen los principales lenguajes que existen para representar modelos de simulación.

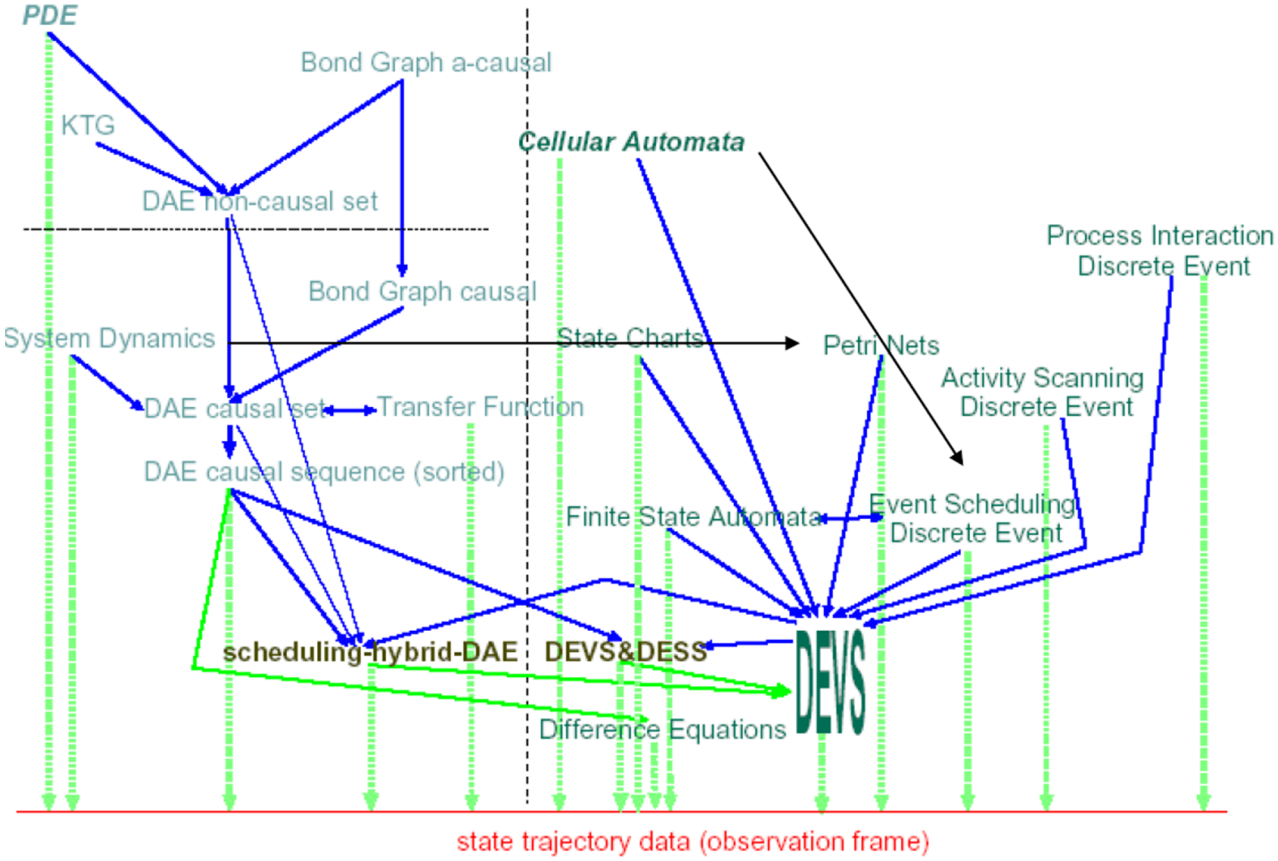
En esta sección vamos a analizar únicamente dos lenguajes, DEVS y Redes de Petri.

6.1. DEVS

Discrete Event System Specification es un lenguaje de modelado matemático que tiene bastantes similitudes con SDL.

El lenguaje fue desarrollado por Bernard P. Zeigler y presentado en público en 1976 (Zeigler, Praehofer y Kim, 2000). Permite representar sistemas híbridos y en principio cualquier sistema puede representase usando DEVS (Vangheluwe, 2000); solamente hay que encontrar sistemas de transformación comunes, como el que se propone aquí para transformar SDL en DEVS (Fonseca i Casas, 2015). Ver figura 27.

Figura 27. Trayectorias de transformación entre diferentes sistemas formales. Se resalta en el grafo la compatibilidad de DEVS



Los sistemas dinámicos en DEVS se definen a partir de una tupla como la que se presenta a continuación:

$S = <T, \quad X, \quad Y, \quad \Omega, \quad Q, \quad \Delta, \quad \Lambda>$, donde:

- T: base del tiempo
- X: conjunto de valores de entrada
- Y: conjunto de valores de salida
- Ω: conjunto de los segmentos de entrada admisibles; w<t1, t2>X, sobre T
- Q: conjunto de valores de los estados
- $\Delta : \quad Q \quad x \quad \Omega \rightarrow Q$ función de transición global
- $\Lambda : \quad Q \quad x \quad X \rightarrow Y$ función de salida

Dado un segmento de trayectoria de la entrada de la actividad (dentro de un conjunto de segmentos de actividad permisible de entrada), se genera un valor inicial del estado (dentro del conjunto de valores de los estados Q) correspondiente en el momento inicial de la actividad de este segmento.

La función de transición usa estos elementos como argumento y calcula el nuevo valor de estado, correspondiente al tiempo final de la entrada del segmento de la actividad.

Finalmente, la salida del sistema es un reflejo del estado (calculado a través de la función de salida).

A partir de estos sistemas se pueden definir los modelos que serán los que nos interesarán para especificar la estructura y el comportamiento de nuestros sistemas.

Un modelo DEVS sigue la siguiente tupla:

M=<X, S, Y, δint, δext, λ, ta>, donde:

- X: conjunto de valores de las entradas
- S: conjunto de valores de los estados
- Y: conjunto de valores de las salidas
- δint: función de transición interna
- $\delta int : S \rightarrow S$
- δext: función de transición externa
- $\delta ext \rightarrow QxX \rightarrow S$, donde
- $Q = \{(s, e) | s \rightarrow S, 0 \leq e \leq ta(s)\}$ es el conjunto de estados total
- e es el tiempo transcurrido desde la última transición
- λ: función de salida: $\lambda : S \rightarrow Y$
- ta: es la función de avance del tiempo
- $ta : S \rightarrow R + 0 \infty$

Dos elementos importantes son la función de transición interna y externa dado que definen la estructura dinámica del modelo.

Transición interna

En algún instante de tiempo el sistema llega al estado s.

- Se mantendrá en ese estado un tiempo ta(s) mientras no reciba ningún evento externo.

- Cuando el tiempo e iguala a ta(s), el sistema produce un evento de salida λ(s) y cambia al estado $s'' = \delta int(s)$.

Transición externa

En algún instante de tiempo el sistema llega al estado s.

- Antes de igualar e a ta(s) llega un evento externo con valor x, es decir el sistema está en el estado (s,e) donde e ≤ta(s).

- El sistema cambia al estado $s'' = \delta ext(s, e, x)$, pero no genera ningún evento de salida.

Algunos comentarios a tener en cuenta:

- Ta(s) puede ser una función que devuelva cualquier número real positivo más el infinito y el 0.

- Si ta(s) es 0 entonces s es un estado transitorio.

- Si ta(s)=∞ entonces es un estado pasivo, en este caso solo se puede salir del estado con un evento externo.

6.1.1. Ejemplo: el procesador

Tenemos un procesador al que le llegan trabajos para desarrollar. Cada trabajo tiene un tiempo de proceso conocido. Una vez transcurrido este tiempo de proceso, se genera un evento que muestra el resultado del trabajo (una función y(..) conocida del trabajo concreto).

Si cuando llega un evento el procesador está trabajando, este se ignora.

Con esta descripción, un modelo DEVS que representara este sistema sería:

M=<X, S, Y, δint, δext, λ, ta> donde:

- X={trabajo1, trabajo2,.., trabajon}

- S={trabajo1, trabajo2,.., trabajon}U[Ø]xR+0∞
- Y={y(trabajo1), y(trabajo2),.. Y(trabajon)}
- δint(trabajo,σ)=(Ø, ∞)
- δext(trabajo,σ,e,x)=
- (x,tp,(x)) si trabajo= Ø
- (trabajo, σ-e) al contrario
- λ(trabajo, σ)=y(trabajo)
- Ta(trabajo, σ)= σ

6.1.2. Ejemplo, la cola FIFO

Tenemos una cola de capacidad infinita a la que llegan trabajos para ser guardados en ella (representados a través de eventos) y un evento especial «ready» que indica que podemos transmitir el primer trabajo de la cola.

La transmisión de este trabajo se hace a través de un evento de salida con este trabajo. Se supone que la cola se demora un tiempo 0 en transmitir este trabajo.

Con esta información el modelo DEVS sería:

M=<X, S, Y, δint, δext, λ, ta>, donde:

- X={trabajo1, trabajo2,.., trabajon}U{„ready”} = JU{„ready”}
- S=J+U[Ø]x{o;∞}
- Y={y(trabajo1), y(trabajo2),.. Y(trabajon)}
- δint(q·trabajo,σ)=(q, ∞)
- δext(q,σ,e,x)=
- (x·q, ∞) si x #J
- (q,0) sino
- λ(q·trabajo, σ)=trabajo

- Ta(trabajo, σ)= σ

Como puede verse en los dos ejemplos, puede darse un caso especial que viene dado por la indefinición provocada por la recepción de un evento externo en el mismo instante que estaba previsto un evento interno. Esto se denomina conflicto de simultaneidad.

Para resolver este problema, DEVS propone la definición de una función de transición confluente y extiende el formalismo a lo que se denomina Parallel-DEVS. Para entender mejor este problema y cómo se resuelve, se puede consultar Zeigler, Praehofer y Kim (2000).

6.1.3. Modularidad en DEVS

Finalmente, vamos a mencionar cómo en DEVS se puede definir la estructura del modelo. Como hemos visto en la sección anterior, con DEVS hemos sido capaces de definir el comportamiento de los diferentes elementos del modelo, de forma similar a como en SDL definíamos los procesos de los AGENTES. Ahora vamos a ver cómo con DEVS se puede definir también la estructura del modelo que dará lugar a la típica descomposición jerárquica de elementos.

Para hacer esto se trabaja con lo que en DEVS se denomina modelos de acoplamiento.

Existen dos tipos de acoplamiento:

- Acoplamiento modular: interacción entre componentes únicamente a través de las entradas y salidas de los mismos.

- Acoplamiento no modular: interacción a través de los estados.

El acoplamiento modular es el que permite establecer fácilmente la jerarquización necesaria para establecer la estructura; para hacerlo, hay que definir la estructura de un modelo acoplado tal como se expresa a continuación:

N=<Xn, Yn,D, {Md}, {Id}, {Z,t,d}, Select>, donde

- X es el conjunto de valores de entrada.

- Y es el conjunto de valores de salida.

- D es el conjunto de referencias a los componentes per cada d.

- Md es una estructura que define un modelo DEVS.

- Md=<Xd, Sd, Yd, δint d, δext d, λd, tad> es un modelo DEVS.

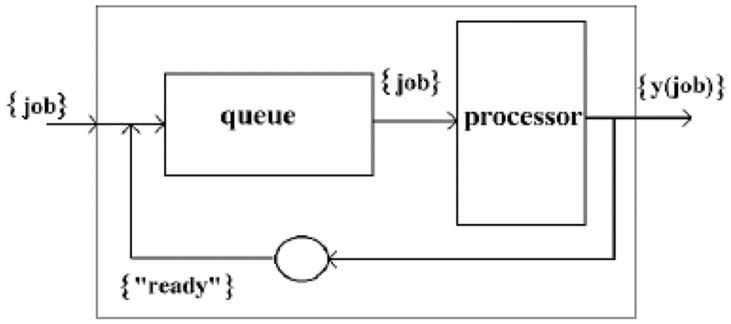
- Para cada $d \rightarrow DU\{N,,\}$ Id es el conjunto de influentes sobre $d: /d \subseteq DU\{N\}, d \notin Id$

Para cada i ∈Id, Zi,d es la función de traducción de salida, donde

- Zt,,d:XN◊Xd, si i=N
- Zt,,d:Yi◊YNsi d=N
- Zt,,d:Yi◊Xd, si d≠N i i≠N
- Select: 2D◊D t.q. Select(E)∈E, función de transición confluente (desempate) en el caso de eventos simultáneos

El acoplamiento puede representarse gráficamente, tal como se muestra en la figura 28.

Figura 28. Modelo acoplado para el procesador y la cola.



6.2. Redes de Petri

Las redes de Petri fueron propuestas por Karl Adam Petri en 1962, en su trabajo de tesis doctoral que tenía por título «Kommunikation mit Automaten». No es el objetivo de esta sección detallar el comportamiento de una red de Petri, pero sí dar indicaciones de cómo funcionan para que el estudiante pueda completar esta información (Cabasino, Giua y Seatzu, 2013). Al igual que DEVS permite trabajar con sistemas híbridos, simplemente se tendría que especificar que las marcas están definidas sobre el cuerpo de los números reales.

Las redes de Petri han sido estandarizadas, como lo es SDL; en la siguiente tabla se detallan las principales etapas seguidas en este proceso de estandarización.

2011-02-14	**Standard published: ISO/IEC 15909-2:2011 Systems and software engineering - High-level Petri nets - Part 2: Transfer format. Available from ISO [Editor's Announcement]**

2005-06-23	New Working Draft of ISO/IEC 15909-2 Systems and software engineering - High-level Petri nets - Part 2: Transfer format submitted for a combined ISO/IEC SC7 WD/CD registration and CD ballot. Comments welcomed - formal or otherwise. [Editor's Announcement | ISO/IEC 15909-2 WD (Version 0.9.0)]
2004-12-02	Standard published: ISO/IEC 15909-1:2004 Systems and software engineering - High-level Petri nets - Part 1: Concepts, definitions and graphical notation. Available from ISO.
2003-02-27	International Standard Ballotof ISO/IEC 15909-1. [BallotZIP.]
2002-11-22	Combined WD/CD Registration/FCD ballotfor ISO/IEC15909-1. [FCDPDF, DispositionRTF.]
2002-10-21	Great news: Green Light for High-level Petri Net Standard. [Letter Ballot Summary PDF.]
2002-09-11	Notificationof ballot for reinstatement of Standard.
2002-05-12	Standard unintentionally cancelledduring Draft International Standard (DIS) phase!
2000-10-28	Final Committee Draft Version 4.7.1 published. [Editor's Announcement| FCD]
1998-01-28	Letter Ballot Summaryof the Standard.
1997-10-02	Draft standard Version 3.4 published.
1997-08-14	Reply to Commentson Draft Petri net Standard V2.1.
1997-01-17	JTC1 News: Standard will not be Cancelled.
1996-12-13	Resolution 8.4, 10th Plenary Meeting of ISO/IEC JTC 1.
1996-11-29	Australian Responseto Cancellation Proposal.
1995-08-14	Petri net Subdivision Proposal.

Una red de Petri (RdP) es un grafo dirigido, ponderado y bipartido. Tiene dos tipos de nodos:

- Lugares: representados por círculos.
- Transiciones: representados por rectángulos.

Los arcos permiten unir Lugares con Transiciones, pero nunca Lugares o Transiciones entre sí (es un grafo bipartido).

Una RdP puede representarse a partir de una tupla, tal como se muestra a continuación:

$RdP=(P,T,A,W,M_0)$

donde

- $P=\{T_1, T_2, T_3,..,T_n\}$: cjt de nodos de tip o Lugar.

- $A \subset (PxT) \cup (TxP)$: cjt de arcos de la red, subconjunto del producto cartesiano de todos los nodos P y T.

- $W = a \rightarrow \{1, 2, 3, \ldots\}$: Peso asociado a cada arco.

- $M_0 = P_i \rightarrow \{1, 2, 3, \ldots\} \forall$ Pi: número de marcas iniciales en cada nodo tipo Lugar (marcado inicial)

- $E(T_j) = \{P_i \in P, (P_j, T_j) \in A\}$: : cjt de Lugares de entrada para la transición T_j.

- $S(T_j) = \{P_i \in, (T_j, P_i) \in A\}$: cjt de Lugares de salida para la transición T_j.

Vamos a ver el funcionamiento de una RdP y cómo se consigue, a través de ella, representar la estructura y el comportamiento de un modelo tal como hemos hecho con SDL.

En una RdP únicamente tenemos un tipo de diagrama y es por ello que tanto la estructura (descomposición jerárquica del modelo) como su comportamiento se representan de modo igual. En la figura 29 tenemos un ejemplo de RdP para un modelo que representa un sistema de colas con dos servidores idénticos.

Vamos a ver cómo funciona la RdP a partir de este ejemplo. En un modelo de colas con dos servidores idénticos, los elementos llegan al sistema siguiendo una pauta conocida. Esta pauta estará representada por una Transición que permitirá introducir el único elemento dinámico de la RdP, el *token* o marca. Esta marca se representa usualmente como un número o un punto en los nodos de tipo Lugar (los círculos). En el ejemplo (ver figura 29), la primera Transición no tiene ningún arco incidente, por tanto, no tiene ninguna condición que limite cuándo la Transición será cierta y, por tanto, generará una marca que viajará por todos sus arcos de salida hacia los lugares de destino.

En el ejemplo, de esta primera Transición se generará una marca que viajará al primer lugar. Hay que remarcar que el número de marcas que se generan en cada arco viene indicado por el peso del arco. En este caso, como no hay ningún valor especificado, se sobreentiende que el valor es 1 (el valor por defecto), pero no tendría por qué ser así y se podría especificar cualquier otro peso, lo que implicaría una generación de marcas diferente.

La segunda Transición sí que tiene un arco de entrada (de hecho, tiene dos), que tiene como peso 1 (dado que no se especifica ningún valor). Esto quiere decir que ahora sí que tenemos una condición para que la Transición pueda ejecutarse; necesitamos como mínimo una marca en el lugar del que proviene cada uno de los dos arcos para poder «consumirlo». En el ejemplo tenemos

que un arco tiene marca en su lugar de origen (tiene dos marcas), mientras que el otro no. Dado que los dos arcos no cumplen la condición, la Transición no se podrá ejecutar.

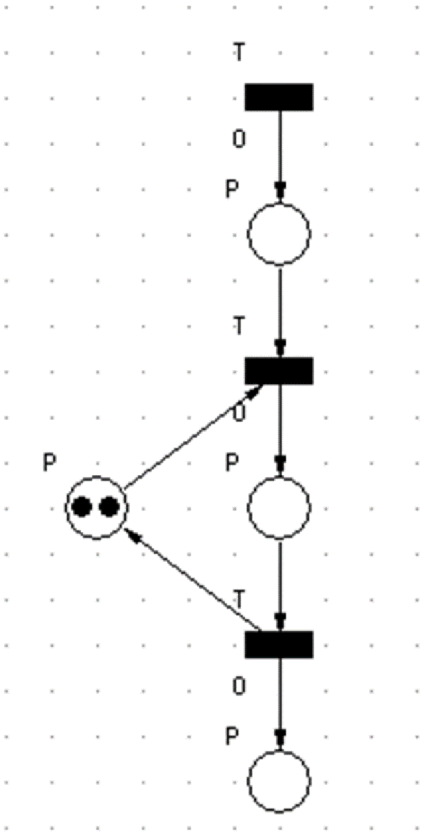
Como se puede ver, las reglas que permiten representar la parte dinámica de una RdP, así como su estructura a través del grafo, son muy simples, pero permiten representar una gran cantidad de situaciones.

En la figura 29, las dos marcas que están en el lugar 3 representan el número de servidores libres que tenemos en el sistema, inicialmente 2. Como se puede ver, si las dos marcas han sido consumidas porque ha llegado alguna marca a través de la primera Transición (que representa la llegada de elementos al sistema), las marcas (piezas para procesar) se acumularán en el primer lugar, que representa la cola de entrada.

Con esta explicación podemos ver que en esta RdP los lugares y sus transiciones tienen la siguiente semántica:

- T1: Llegada de elementos al sistema.
- T2: Inicio de servicio.
- T3: Fin de servicio.
- P1: Cola de entrada.
- P2: Elementos en servicio.
- P3: Servidores libres.
- P4: Elementos servidos.

Figura 29. RdP de un modelo de colas con dos servidores idénticos.

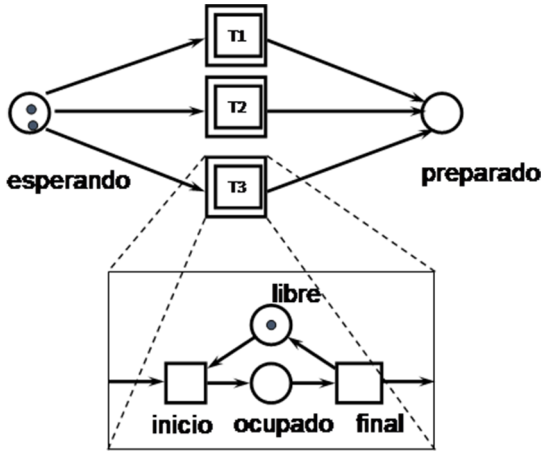


Existe infinidad de RdP que permiten especificar los sistemas de forma más modular, como las RdP con colores, las temporales, etc. Para los modelos con los que trabajaremos en este curso, usualmente las RdP temporales son las más interesantes. En este caso, simplemente hay que recordar que las transacciones pueden tener asociadas, además de las condiciones propias de los arcos de entrada, distribuciones de probabilidad que representan el tipo necesario para que la Transición se ejecute.

6.2.1. Modularidad en las RdP

Si el modelo con el que trabajamos es grande, la RdP acostumbra a crecer de forma que puede que no sea comprensible. Para resolver este problema, las RdP presentan la capacidad de modularizarse, es decir, de representarse a partir de módulos que representan sus transiciones. En la figura 30 se puede apreciar un ejemplo de modularidad para una RdP concreta.

Figura 30. Modularidad en una RdP.



Como puede verse, las transiciones pueden representar RdP completas, simplemente hay que asegurar la coherencia entre el número y tipo de arcos de entrada y salida que tengamos.

Bibliografía

Cabasino, M. P.; Giua, A.; Seatzu, C. (2013). «Introduction to petri nets». *Lecture Notes in Control and Information Science* (vol. 433, págs. 191-211). Springer.

Doldi, L. (2001). *SDL illustrated. Visually design executable models*. Toulose: TMSO Multimedia.

Fonseca i Casas, P. (2010). «Using specification and description language to define and implement discrete simulation models». En: *Actas de la 2010 Summer Computer Simulation Conference, SCSC '10* (núm. 1, págs. 419-426).

Fonseca i Casas, P. (2014a). *Formal languages for computer simulation: Transdisciplinary models and applications*. Hershey, PA: IGI Global. DOI: 10.4018/978-1-4666-4369-7.

Fonseca i Casas, P. (2014b, mayo). «Using specification and description language to formalize multiagent systems». *Applied Artificial Intelligence* (vol. 28, núm. 5, págs. 504-531).

Fonseca i Casas, P. (2015, marzo). «Transforming classic discrete event system specification models to Specification and Description Language». *Simulation* (vol. 91, págs. 249-264).

ITU-T (2011). *Specification and Description Language – Overview of SDL-2010*.

PragmaDev Studio [en línea]. [Fecha de consulta: 9 de enero de 2016]. PragmaDev SARL, 2016. <http://www.pragmadev.com/product/index.html>

Vangheluwe, H. L. M. (2000). «DEVS as a common denominator for multi-formalism hybrid systems modelling». En: *Actas del CACSD 2000. IEEE International Symposium on Computer-Aided Control System Design* (Cat. núm. 00TH8537).

Wilensky, U. (2016). *NetLogo* [en línea]. [Fecha de consulta: 23 de julio de 2017]. <https://ccl.northwestern.edu/netlogo/>

Zeigler, B. P.; Praehofer, H.; Kim, T. G. (2000). *Theory of modelling and simulation: Integrating discrete event and continuous complex dynamic systems*. San Diego: Academic Press.