Generación de procesos markovianos para la evaluación de sistemas complejos

Héctor Cancela[*]
Investigación Operativa, IN.CO.
Facultad Ingeniería – J. Herrera y Reissig 565
CP 11300 – Montevideo, Uruguay
cancela@fing.edu.uy

Gerardo Rubino
IRISA/INRIA
Campus de Beaulieu
F–35042 Rennes Cédex, France
rubino@irisa.fr

Resumen

Este trabajo se sitúa en el contexto de la evaluación de la seguridad de funcionamiento de sistemas complejos (especialmente informáticos y de comunicaciones). Para el cálculo de las medidas de la seguridad de funcionamiento (confiabilidad, disponibilidad, mantenibilidad, etc.) se emplean usualmente procesos estocásticos (típicamente markovianos) como modelos de los sistemas reales. Un problema fundamental que aparece con frecuencia es el gran tamaño del espacio de estado de estos modelos, lo que hace imposible su especificación directa.

En este trabajo describimos un método a la vez simple y potente de construcción de modelos, llamado formalismo de "bolas y urnas" (balls and buckets). Presentamos una biblioteca de clases C++ implementada en dos niveles: un núcleo de generación de cadenas de Markov en tiempo continuo, incluyendo la eliminación de estados instantáneos, y una interface que realiza la interpretación para el núcleo de las descripciones dadas en el formalismo de bolas y urnas.

En el artículo mostramos como utilizar esta biblioteca para generar los procesos markovianos correspondientes a descripciones realizadas a un nivel más alto, entre otros las redes de Petri estocásticas y los sistemas de filas de espera, utilizando el formalismo de bolas y urnas como un "lenguaje ensamblador".

Discutimos por último la potencia y los límites de este enfoque, ilustrando el análisis con la descripción de algunas clases específicas de sistemas y los modelos markovianos correspondientes obtenidos utilizando la biblioteca de generación antes presentada.

Palabras clave: *generación de procesos markovianos, modelado de sistemas, seguridad de funcionamiento.*

Abstract

As computing and communication systems become more complex, their evaluation calls for the use of increasingly sophisticated tools and techniques. Stochastic processes, typically markovian ones, are usually employed to model real systems in order to calculate dependability measures (reliability, availability, maintainability, etc.). A fundamental problem is the large size of the state space of these models, which precludes their direct specification.

[*]El trabajo de Héctor Cancela fue apoyado por becas del Ministère des Affaires Étrangères, France, y por el CONICYT y el PEDECIBA Informática, Uruguay

In this work we describe a simple and powerful model construction framework, based upon the concept of "balls and buckets". We present a C++ library, implemented in two levels: a nucleus that generates continuous time Markov processes, eliminating instantaneous states, and an interface which interprets a balls and buckets description in terms of the nucleus' primitives.

We also show how to use this library to generate Markov processes that correspond to higher level descriptions, stochastic Petri nets and queueing networks among others. To do this, we use the balls and buckets formalism as an "assembler" language.

Finally, we discuss the advantages and the limits of this approach, illustrating with the description of some specific computer systems architectures and the corresponding markovian models generated by the library.

Keywords: *dependability, Markov processes generation, complex systems modeling.*

1 Introducción

Las técnicas de modelado y de evaluación de los sistemas informáticos y de comunicación han recibido una atención creciente en los últimos 20 años. Numerosas actividades dependen del funcionamiento de estos sistemas y crean la necesidad de diseñar arquitecturas cada vez más potentes. En particular es el caso de los sistemas llamados críticos, a grandes riesgos humanos o económicos (sistemas de control de centrales nucleares, sistemas de navegación espacial o aérea, etc.). Distintos conceptos de base han sido desarrollados para analizar estos sistemas, en particular el de *seguridad de funcionamiento*, introducido en [Mey80], de difundida aplicación en la actualidad. Tomamos de [Lap92] la siguiente definición:

- **seguridad de funcionamiento** (de un sistema): es la propiedad que permite a los usuarios del sistema depositar una confianza justificada en el servicio que éste les proporciona.

Las principales medidas de la seguridad de funcionamiento son:

- **la confiabilidad**: medida del suministro continuo de un servicio correcto, o lo que es equivalente, del tiempo hasta un fallo;
- **la disponibilidad**: medida del suministro de un servicio apropiado respecto a la alternancia servicio apropiado – servicio inapropiado;
- **la mantenibilidad**: medida del tiempo hasta la reparación a partir del último fallo acaecido, o, lo que es lo mismo, del suministro continuo de un servicio inapropiado;
- **la performabilidad**: medida combinada performance – seguridad de funcionamiento, en los sistemas con múltiples niveles de performance.

Para calcular estas medidas los sistemas son modelados como procesos estocásticos, típicamente en un cuadro markoviano. Para agregar consideraciones de performance es posible utilizar procesos con recompensas. Varias técnicas han sido desarrolladas para realizar cálculos sobre estos modelos. El problema principal es que el tamaño del espacio de estado de los procesos crece proporcionalmente a la complejidad y el nivel de detalle del modelo. lo que hace imposible que el usuario especifique directamente el proceso subyacente. Esta tarea requiere interfaces de alto nivel, que ofrezcan una representación compacta del sistema estudiado. Lo ideal es que estas interfaces permitan al usuario describir el sistema estudiado a través de los mismos conceptos usados, por ejemplo, para diseñarlo.

Es posible entonces descomponer la estructura de una herramienta de modelado en (i) un módulo de interface para la descripción de los modelos y la recuperación de los resultados, (ii) un módulo de construcción de los objetos matemáticos que van a representar el comportamiento del sistema, y (iii) un módulo de cálculo que permita obtener las medidas deseadas a partir de los objetos construídos en (ii).

Presentamos en este trabajo una biblioteca que puede ser utilizada como base para la construcción de una herramienta de modelado. Esta biblioteca implementa las tareas correspondientes al segundo módulo, generando una descripción que puede ser utilizada por un módulo de cálculo, y facilitando la tarea a realizar por el módulo de interface.

La organización del texto es la siguiente. En la sección 2 presentamos el formalismo llamado de *bolas y urnas* (balls and buckets). En la sección 3 discutimos la biblioteca de generación y su organización en dos niveles, que consisten en un núcleo de generación y una interface de interpretación del formalismo adoptado. La sección 4 muestra como utilizar esta biblioteca para generar los procesos markovianos correspondientes a descripciones realizadas en los formalismos de redes de Petri estocásticas, sistemas de filas de espera, y modelos MRM. Presentamos luego en la sección 5 un ejemplo de descripción de una familia de modelos, ilustrándolo con la descripción de algunas clases específicas de sistemas y los modelos markovianos correspondientes obtenidos utilizando la biblioteca de generación antes presentada. La sección 6 está dedicada a las conclusiones.

2 Formalismo de “bolas y urnas”

2.1 Descripción

El formalismo que llamamos de *bolas y urnas* (“ball and buckets” en [Ste91]) ha sido propuesto para especificar de una forma compacta procesos de Markov de tiempo continuo, a través de un conjunto reducido de reglas muy sencillas que permiten construir automáticamente el espacio de estado y el generador infinitesimal. El estado del sistema es representado por un conjunto de L urnas, cada una de las cuales contiene una cantidad n_i de bolas, con $n_i \in [0, nmax_i]$, $i = 1, 2, \ldots, L$. Las bolas pueden transitar entre las distintas urnas. El movimiento de una bola de una urna i hacia una urna j es llamado *transferencia* de i a j, y solo puede tener lugar si $n_i > 0$ y $n_j < nmax_j$. Las transferencias se efectúan siguiendo una ley exponencial con tasas que pueden depender del estado global del sistema, es decir del número de bolas en cada una de las urnas; estas tasas pueden ser eventualmente infinitas. Una transferencia puede ser seguida de una modificación arbitraria del contenido de las urnas que se efectúa de forma instantánea. Una *transición* es un cambio del estado del sistema. Las transiciones consideradas son entonces las transferencias y las modificaciones instantáneas. Una transición con tasa de franqueo infinita recibe el nombre de *transición instantánea* (es el caso de las modificaciones instantáneas y de las transferencias con tasa infinita). Un estado del sistema que no admite ninguna transición instantánea es un estado *tangible*. Recordamos que bajo algunas condiciones muy generales (no existencia de una clase recurrente formada exclusivamente por estados instantáneos (no tangibles), ver [Cia93]) es posible eliminar las transiciones instantáneas de un proceso markoviano obteniendo otro proceso que es estocásticamente equivalente al primero.

Para describir un sistema con este formalismo, es necesario dar el número de urnas L, el número máximo de bolas posibles $nmax_i$ para todas las urnas i (en principio consideraremos que para todo i, $nmax_i < \infty$; discutiremos más adelante esta restricción), la lista de las transferencias $i \to j$ posibles y sus tasas en función del estado global del sistema, tasas que notaremos $\lambda_{ij}(\vec{n})$. Por último, es necesario también especificar una función que proporcione las modificaciones instantáneas posibles para cada estado, y sus probabilidades respectivas.

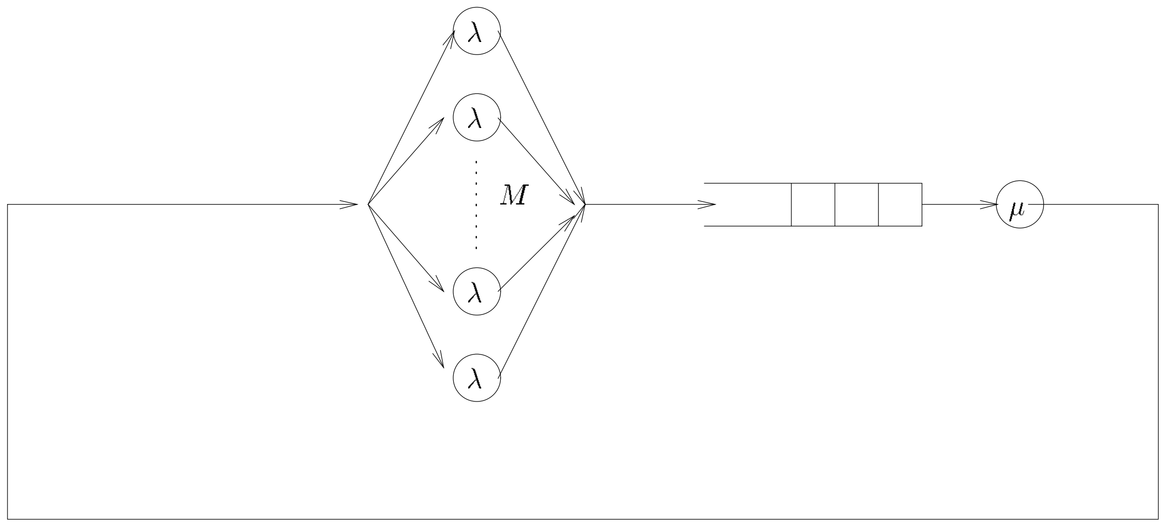

Figura 1 : Modelo “K-out-of-M”

Veamos un ejemplo de descripción de un sistema en este formalismo. En la figura 1 podemos ver la representación gráfica de un sistema de filas de espera que corresponde a un sistema “K-out-of-M” con un solo reparador. Un sistema de este tipo está compuesto por M componentes idénticos de comportamiento independiente, que pueden fallar con una duración de vida de distribución exponencial de tasa λ. Un reparador emplea un tiempo exponencial de tasa μ para reparar estos componentes. Para que el sistema esté en estado operativo, es necesario tener por lo menos K componentes en funcionamiento.

Presentamos a continuación un modelo *bolas y urnas* equivalente. Tenemos dos urnas, las bolas de la primer urna corresponden a los componentes en funcionamiento, y aquellas de la segunda urna a los componentes en reparación. Las transferencias corresponden a los eventos de rotura y reparación de componentes. En este ejemplo el número total de bolas se conserva, lo que no siempre es el caso.

- $L = 2$;
- $nmax_i = M, i = 1, 2$;
- rotura de un componente: $(a, b) \xrightarrow{a\lambda} (a - 1, b + 1), a > 0$;
- reparación de un componente: $(a, b) \xrightarrow{\mu} (a + 1, b - 1), b > 0$;
- ninguna transición instantánea;
- estado inicial: $(M, 0)$.

Veamos otro ejemplo que incorpora el uso de transiciones instantáneas. Supongamos un servidor que recibe solicitudes con tasa λ. Cuando una solicitud llega y el servidor está disponible, se comienzan dos tareas en paralelo que demoran un tiempo aleatorio exponencial de tasas μ_1 y μ_2 respectivamente. El servidor no acepta ninguna otra solicitud hasta que ambas tareas hayan sido finalizadas (hay una operación de sincronización entre ambas tareas). El siguiente es un modelo *bolas y urnas* que describe este sistema:

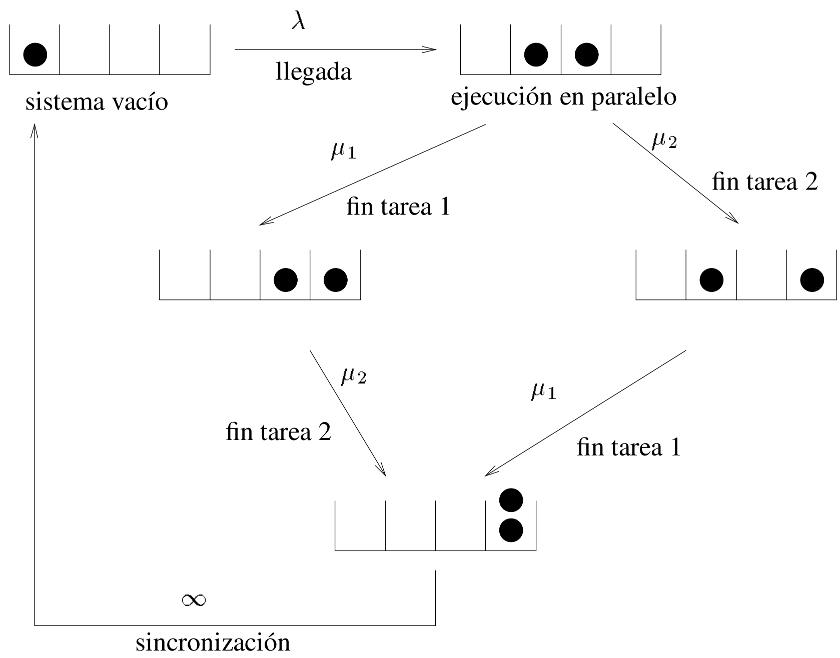

Figura 2 : Modelo *bolas y urnas* de un servidor con sincronización

- $L = 4$;
- $nmax_i = 1, i = 1, 2, 3$; $nmax_4 = 2$;
- llegada de una solicitud: $(1, 0, 0, 0) \xrightarrow{\lambda} (0, 1, 1, 0)$;
- finalización de la primer tarea: $(0, 1, a, b) \xrightarrow{\mu_1} (0, 0, a, b + 1)$;
- finalización de la segunda tarea: $(0, a, 1, b) \xrightarrow{\mu_2} (0, a, 0, b + 1)$;
- sincronización de ambas tareas, y retorno al estado de disponibilidad del servidor: $(0, 0, 0, 2) \xrightarrow{\infty} (1, 0, 0, 0)$.
- estado inicial: $(1, 0, 0, 0)$

La figura 2 representa los estados y las transiciones generadas por este modelo. El espacio de estado $\mathcal{E}$ de este modelo es

$$\mathcal{E} = \{(1, 0, 0, 0), (0, 1, 1, 0), (0, 1, 0, 1), (0, 0, 1, 1), (0, 0, 0, 2)\}.$$

El estado inicial $(1, 0, 0, 0)$ corresponde a un sistema vacío, en el que el servidor está esperando la llegada de una solicitud. El estado $(0, 1, 1, 0)$ representa la llegada de una solicitud al servidor y la subsecuente activación de las dos tareas en paralelo. Los estados $(0, 1, 0, 1)$ y $(0, 0, 1, 1)$ corresponden a la situación en la cual una de las dos tareas ya ha sido terminada, pero la otra no. El último estado, $(0, 0, 0, 2)$ es no tangible. Se llega a este estado cuando ambas tareas han sido finalizadas, y por lo tanto solo queda realizar la operación de sincronización (que suponemos de duración nula) para volver al estado inicial del sistema. El proceso de Markov subyacente X puede ser caracterizado por su espacio de estado

$$\mathcal{S} = \{(1,0,0,0), (0,1,1,0), (0,1,0,1), (0,0,1,1)\}$$

y su generador infinitesimal M, donde

$$M = \begin{array}{c} \\ (1,0,0,0) \\ (0,1,1,0) \\ (0,1,0,1) \\ (0,0,1,1) \end{array} \begin{array}{c} \begin{array}{cccc} (1,0,0,0) & (0,1,1,0) & (0,1,0,1) & (0,0,1,1) \end{array} \\ \begin{pmatrix} -\lambda & \lambda & 0 & 0 \\ 0 & -(\mu_1+\mu_2) & \mu_2 & \mu_1 \\ \mu_1 & 0 & -\mu_1 & 0 \\ \mu_2 & 0 & 0 & -\mu_2 \end{pmatrix} \end{array}$$

Una descripción bolas y urnas genérica se define por $(L, \overrightarrow{nmax}, \vec{e}, \lambda, \Gamma)$, donde:

- $L \geq 1$ es el número de urnas;
- $\overrightarrow{nmax} \in \mathbb{N}^L$ es el vector de las capacidades máximas de cada urna;
- $\vec{e} \in \mathbb{N}^L, 0 \leq e_i \leq nmax_i \; \forall i \in 1..L$, es el estado inicial;
- $\lambda \in (\mathbb{N}^L \to \overline{\mathbb{R}}^+)^{L \times L}$ es la matriz de las tasas de las transferencias (en función del estado actual). Para todos $\vec{n}$, i, j, tales que $0 < n_i$ y $n_j < nmax$,

  $$\lambda_{ij}(\vec{n}) = \text{tasa de transferencia de una bola de la urna } i \text{ a la urna } j;$$

  Si $n_i = 0$ o si $n_j = nmax_j$, por definición $\lambda_{ij}(\vec{n}) = 0$.

- $\Gamma \in \left(\mathbb{N}^L \to (\mathbb{N}^L \times [0,1])^*\right)^{L \times L}$ es la matriz de las funciones que dan las modificaciones instantáneas:

  $$\Gamma_{ij}(\vec{n}) = \begin{cases} \emptyset & \text{si } \vec{n} \text{ es un estado tangible;} \\ ((\overrightarrow{m_1}, p_1), \ldots, (\overrightarrow{m_H}, p_H)) & \text{donde } \overrightarrow{m_1}, \ldots, \overrightarrow{m_H} \text{ son los distintos estados accesibles a partir del estado } \vec{n} \text{ (alcanzado mediante una transferencia de la urna } i \text{ a la urna } j\text{). La probabilidad de que la transición sea hacia } \overrightarrow{m_h} \text{ es } p_h. \end{cases}$$

En las definiciones de λ y Γ, preferimos utilizar $\mathbb{N}^L$ y no el subespacio de los estados accesibles porque, en la práctica, dicho subespacio es en general desconocido; es claro que el utilizar $\mathbb{N}^L$ no afecta el resultado.

Podemos observar que este formalismo es de potencia equivalente (en el sentido de la clase de modelos descritos) al formalismo de los procesos markovianos finitos de tiempo continuo. En efecto, cualquier proceso markoviano de espacio de estado finito y tiempo continuo puede ser descrito como un modelo *bolas y urnas*: si el proceso $X(t)$ tiene un espacio de estado $\mathcal{S} = \{a_1, a_2, \ldots, a_L\}$ y un generador infinitesimal $Q = (q_{ij})$, alcanza con tomar L urnas, transferencias entre todos los pares de urnas i y j con tasas q_{ij}, y ninguna transición instantánea. Una sola bola se desplazará entre las urnas. Recíprocamente, un modelo bolas y urnas es por su definición un proceso markoviano donde algunas transiciones se efectúan a una tasa infinita. Es posible eliminar estas transiciones y obtener un proceso markoviano normalizado, que resulta equivalente al anterior desde el punto de vista de las medidas de interés para nuestro trabajo.

Si se generaliza el formalismo permitiendo utilizar urnas con un número no acotado de bolas, tenemos entonces la clase de los procesos markovianos de espacio de estado numerable y tiempo continuo. Resulta claro que los métodos clásicos de cálculo, que se basan en el conocimiento previo del espacio de estado y el generador instantáneo del proceso en cuestión, son inaplicables en este caso. Sin embargo, es posible utilizar métodos adaptados a distintas medidas que sólo utilizan una parte de dicho espacio de estado. Puede entonces resultar interesante incorporar esta generalización a una herramienta de modelado, si se agregan dichos métodos en el nivel adecuado.

2.2 Comparación con otros formalismos

Podemos agrupar la gran mayoría de los formalismos utilizados para la descripción de modelos markovianos de seguridad de funcionamiento en las siguientes familias: redes de Petri estocásticas, sistemas de filas de espera, formalismos lógicos, lenguajes especializados. A continuación discutiremos brevemente las características de estos formalismos, y sus ventajas y desventajas en relación al de *bolas y urnas*.

Las redes de Petri estocásticas son una extensión del formalismo bien conocido de redes de Petri. La diferencia básica consiste en la incorporación de tiempos aleatorios (distribución exponencial) de franqueo de las transiciones habilitadas. Este es un formalismo flexible, de potencia equivalente al de los procesos markovianos de tiempo continuo, que admite una representación gráfica bastante intuitiva. Existen varios métodos para controlar las propiedades estructurales de una red de Petri, lo que resulta interesante para su validación. Para el cálculo de las distintas medidas de interés se recurre en general a la generación de cadenas de Markov equivalentes. El problema fundamental es que las construcciones de base son todas de muy bajo nivel, lo que implica que para modelar un sistema moderadamente complejo aparecen dificultades insalvables en cuanto al tamaño y la complejidad de la descripción correspondiente. Para superar este inconveniente diversas extensiones del formalismo han sido propuestas. Estas extensiones incorporan, típicamente, generalizaciones de las distribuciones de probabilidad de los tiempos de transición y/o la incorporación de funciones arbitrarias de modificación del estado de la red, asociadas a los lugares, a las fichas o a las transiciones. Esto implica la pérdida (o por lo menos el debilitamiento) de las propiedades estructurales controlables en el modelo de base, pero permite realizar descripciones de tamaño razonable. Otros inconvenientes de este formalismo son el hecho de que no resulta fácil adaptar el lenguaje para facilitar su aplicación a un dominio específico, y también el que al cambiar una sola característica esencial del sistema muchas veces es necesario reescribir el modelo a partir de cero. Frente a las redes de Petri, la principal ventaja del formalismo de bolas y urnas consiste en su simplicidad. Si bien es cierto que a cambio se pierde parte de la riqueza de propiedades estructurales asociadas a las redes de Petri más simples, estas propiedades ya no están presentes en la mayoría de las extensiones utilizadas en herramientas de modelado, extensiones necesarias para permitir una representación suficientemente compacta de los sistemas modelados.

Las redes de filas de espera son un formalismo muy potente de modelado de sistemas, de alcance mucho más amplio que los sistemas markovianos. En general, es bastante difícil obtener resultados analíticos sobre las redes de filas de espera genéricas. Sin embargo, distintas clases de redes poseen propiedades que facilitan su estudio, y permiten obtener expresiones analíticas de las medidas de interés, a veces pasando por su conversión en cadenas de Markov de tiempo continuo. Existen también diversas técnicas numéricas iterativas (búsquedas de punto fijo, por ejemplo), y en último caso siempre es posible utilizar la simulación para estimar el comportamiento del sistema. Las redes de filas de espera son especialmente utilizadas en el cuadro de la evaluación de las performances. Los principales inconvenientes de este formalismo son las dificultades que se encuentran para expresar la sincronización, y la pérdida de la solución analítica para numerosas disciplinas de fila encontradas en la realidad.

Varias herramientas emplean lenguajes desarrollados específicamente para la descripción de una clase particular de sistemas, con la finalidad de facilitar la tarea del usuario. El objetivo buscado es describir en forma concisa sistemas complejos. Las ventajas de este enfoque consisten en la posibilidad de aprovechar las características propias de la clase de sistemas modelados para en una primer etapa intentar reducir el número de estados de la cadena de Markov y posteriormente elegir el método de solución más adaptado. El problema principal se encuentra en el hecho de que el usuario está obligado a utilizar las construcciones de alto nivel previstas en el lenguaje, que pueden no ser de una generalidad suficiente para tener en cuenta ciertos aspectos del sistema a modelar. Un enfoque de este tipo es entonces radicalmente distinto al que nosotros proponemos, en el que buscamos un paradigma de "bajo nivel" que actúe como lenguaje ensamblador, siendo de una generalidad suficiente para expresar cualquier tipo de sistema.

Existe una serie de formalismos de descripción basados en formalismos lógicos y formalismos mixtos lógicos-orientados a objetos. La idea básica consiste en especificar a través de reglas la evolución del sistema, es decir el pasaje de un estado al siguiente. En el caso de los formalismos mixtos, las reglas en lugar de modificar el estado global del sistema, se refieren al comportamiento de los objetos individuales que lo integran. Este enfoque posee un nivel de generalidad y una potencia de descripción muy interesantes, y se presta al desarrollo de interfaces apropiadas a distintos dominios o a aplicaciones específicas. Algunos inconvenientes consisten en que el modelo de base es bastante complejo, de uso poco intuitivo, y en particular resulta difícil para el usuario realizar la validación (completitud y coherencia) del conjunto de reglas que describen un modelo.

El formalismo de bolas y urnas ha sido utilizado en la herramienta MARCA ([Ste91]) como lenguaje de especificación de cadenas de Markov. La implementación allí realizada no permite la especificación de transferencias de tasa infinita (una única modificación instantánea puede estar asociada a la finalización de una transferencia). Esto obliga al usuario a tener en cuenta los posibles encadenamientos de transiciones instantáneas e incorporar en la descripción del modelo sus efectos combinados. Aunque MARCA es utilizable no sólo como programa aislado sino también como biblioteca, no incorpora mecanismos simples para su adaptación a otros formalismos distintos del de bolas y urnas. En la biblioteca que presentamos en la siguiente sección, estos aspectos han sido tenidos en cuenta. En lo que se refiere al primer punto, damos al usuario la posibilidad de especificar tasas infinitas en cualquier transición. Este enfoque es más general y permite encadenar automáticamente varias transiciones instantáneas. Por otro lado, la estructura en capas de la herramienta que proponemos tiene precisamente como finalidad la posibilidad de incorporar otros formalismos de descripción de sistemas (en la sección 4 discutimos varios ejemplos). Facilita esta tarea el haber utilizado para la implementación un lenguaje orientado a objetos (C++).

3 Biblioteca de generación

La biblioteca que presentamos está compuesta por un conjunto de clases y métodos, implementados en C++, que a partir de un modelo de seguridad de funcionamiento de un sistema construyen el proceso de Markov a utilizar ulteriormente para el cálculo de las medidas deseadas. La biblioteca está organizada en dos niveles. El primero es un núcleo genérico que no depende del formalismo elegido para describir el sistema. El segundo nivel consiste en una especialización de las clases del núcleo para el caso particular en el que la descripción del sistema se realiza en el formalismo de *bolas y urnas*.

```
ℰ := {e_0}
repetir ∀ estado e_i ∈ ℰ, e_i sin procesar
        repetir ∀ estado e_j tal que existe una transición
            de e_i hacia e_j de tasa λ_ij
                si e_j ∉ ℰ
                       ℰ := ℰ ∪ {e_j}
                fin si
                agregar el valor λ_ij en la entrada (e_i, e_j) de M
        fin repetir
        sumar el valor −∑_{e_j} λ_ij en la entrada (e_i, e_i) de M
        marcar e_i como procesado
fin repetir
```

Figura 3 : Algoritmo de generación para un sistema sin transiciones instantáneas

3.1 Núcleo

El núcleo consiste en un conjunto de clases y métodos genéricos, que no determinan la elección del formalismo a utilizar para la descripción del sistema. Las hipótesis de base son las siguientes. Las transiciones entre los estados del sistema se realizan en tiempo exponencial o nulo. La descripción del sistema incluye una función que proporciona todas las transiciones posibles a partir de un estado y sus tasas asociadas (en el caso de transiciones instantáneas, sus probabilidades). Se considera conocido también el estado inicial del sistema.

La biblioteca construye el conjunto $\mathcal{E}$ de los estados que pueden ser alcanzados a partir del estado inicial y el proceso de Markov subyacente X que se obtiene por eliminación de los estados no tangibles, dado por su espacio de estado $\mathcal{S}$ ($\mathcal{S} \subseteq \mathcal{E}$), su generador infinitesimal Q y su estado inicial.

El algoritmo que hemos elegido construye la matriz de las transiciones del sistema en forma incremental, a través de la búsqueda de los estados accesibles a partir del estado inicial. Esta técnica tiene diversas ventajas. Por ejemplo, solamente los estados que pueden ser alcanzados a partir del estado inicial estarán efectivamente representados en la matriz de transición resultante (lo que representa muchos menos elementos que el producto cartesiano de los estados de los objetos individuales que constituyen el sistema, tomado como espacio de estado en algunos formalismos de modelado).

Supongamos en primera instancia que nuestro sistema no posee transiciones instantáneas. En tal caso, todos los estados son tangibles y el espacio de estado $\mathcal{E}$ del sistema modelado es el mismo que el espacio de estado $\mathcal{S}$ del proceso de Markov subyacente. El algoritmo de la figura 3 construye, a partir del estado inicial e_0, dicho espacio de estado, y almacena en la matriz M, que llamaremos *matriz de transiciones*, el generador infinitesimal Q del proceso de Markov (observar que cada coeficiente de la diagonal de esta matriz es igual al opuesto de la suma de los otros coeficientes de la linea correspondiente).

Cuando tomamos en consideración el caso general de los modelos con transiciones instantáneas el espacio de estado $\mathcal{E}$ contiene los dos subconjuntos $\mathcal{W}$, formado por los estados tangibles, y $\mathcal{V}$, formado por los estados no tangibles. El espacio de estado del proceso markoviano subyacente X es igual a $\mathcal{W}$, y el generador infinitesimal no puede ser calculado directamente. En este caso almacenaremos en la matriz de transiciones M tanto las tasas de las transiciones exponenciales como las probabilidades de transición asociadas a las transiciones instantáneas, para luego utilizar un algoritmo (discutido en los párrafos siguientes)

```
ℰ := {e_0}
𝒲 := {}
repetir ∀ estado e_i ∈ ℰ, e_i sin procesar
        repetir ∀ estado e_j tal que existe una transición
            del estado e_i hacia e_j (de tasa λ_ij o probabilidad p_ij)
                si e_j ∉ ℰ
                        ℰ := ℰ ∪ {e_j}
                fin si
                agregar el valor λ_ij o p_ij en la entrada (e_i, e_j) de M
        fin repetir
        si e_i no posee transiciones instantáneas
                sumar el valor −∑_{e_j} λ_ij en la entrada (e_i, e_i) de M
                𝒲 := 𝒲 ∪ {e_i}
        fin si
        marcar e_i como procesado
fin repetir
```

Figura 4 : Algoritmo de generación para un sistema con transiciones instantáneas

de cálculo del generador infinitesimal del proceso X. En el nuevo algoritmo de generación (figura 4) se distinguen dos casos: si un estado e_i no posee transiciones instantáneas, todas las entradas (e_i, e_j) en la matriz M son tasas de transición y la diagonal (e_i, e_i) es como anteriormente el opuesto a la suma de las otras tasas; cuando un estado e_i posee transiciones instantáneas, las entradas (e_i, e_j) en M son probabilidades de transición. El elemento (e_i, e_i) es entonces la probabilidad de transición instantánea de e_i hacia sí mismo (generalmente igual a 0).

Para eliminar los estados con transiciones instantáneas y calcular el proceso de Markov de tiempo continuo con espacio de estado $\mathcal{S} = \mathcal{W}$ (subconjunto de los estados tangibles del sistema modelado) es necesario calcular el generador infinitesimal Q de las tasas totales de transición entre los estados tangibles.

Sean $i, j \in \mathcal{W}$ dos estados tangibles cualesquiera. La tasa de transición entre i y j, q_{ij}, verifica la siguiente ecuación:

$$q_{ij} = m_{ij} + \sum_{r \in \mathcal{V}} m_{ir} \omega_{rj},$$

donde m_{ij} es la entrada (e_i, e_j) de la matriz M y ω_{rj} es la probabilidad de que el sistema transite del estado r hacia el estado j en un número arbitrario de transiciones siguiendo un camino constituído por estados no tangibles.

Para estudiar el comportamiento de los estados no tangibles, definimos una cadena de Markov Y de espacio de estado $\mathcal{Z} = \mathcal{V} \cup \{j \in \mathcal{W} | \exists r \in \mathcal{V}, m_{rj} > 0\}$ y matriz de transición $P = (p_{rs})$ dada por $p_{rs} = m_{rs}$ cuando $r \in \mathcal{V}, s \in \mathcal{Z}$, $p_{rs} = 0$ cuando $r \in \mathcal{Z} \setminus \mathcal{W}, s \in \mathcal{Z}, s \neq r$ y $p_{rr} = 1$ para $r \in \mathcal{Z} \setminus \mathcal{W}$. Los estados $j \in \mathcal{Z} \setminus \mathcal{V}$ son estados absorbentes de la cadena. El valor ω_{rj} es igual a la probabilidad de absorción por el estado j cuando el estado inicial es r. Se cumple entonces la siguiente relación:

$$\omega_{rj} = p_{rj} + \sum_{s \in \mathcal{V}} p_{rs} \omega_{sj},$$

que puede ser reescrita[1]

$$(I - \tilde{P})\omega_{\star j} = p_{\star j},$$

donde $\tilde{P}$ es la matriz de transicion restringida a los estados transitorios. Como solo el segundo miembro de este sistema lineal depende del estado absorbente j, podemos resolver simultáneamente para todos los estados $j \in \mathcal{Z} \setminus \mathcal{V}$ (sistema lineal con múltiples segundos miembros). De esta manera encontramos los valores ω_{rj} necesarios para calcular el generador infinitesimal de X.

En resumen, el algoritmo a aplicar es el siguiente:

construir la cadena de Markov Y
resolver el sistema lineal con múltiples segundos miembros
$\quad (I - \tilde{P})(\omega_{\star j})_{j \in \mathcal{Z} \setminus \mathcal{V}} = (p_{\star j})_{j \in \mathcal{Z} \setminus \mathcal{V}}$
calcular el generador infinitesimal Q de X:
$\quad q_{ij} = m_{ij} + \sum_{r \in V} m_{ir}\omega_{rj}, \quad \forall i, j \in \mathcal{W}$

Este algoritmo, tal como está planteado, implica el resolver un sistema lineal de tamaño $|\mathcal{V}|$, con $|\mathcal{Z} \setminus \mathcal{V}|$ segundos miembros. El tiempo de ejecución será entonces de orden $|\mathcal{V}|^2 \times max(|\mathcal{V}|, |\mathcal{Z} \setminus \mathcal{V}|)$. Para disminuir este tiempo, descomponemos la cadena de Markov Y en sus componentes fuertemente conexas y aplicamos las fórmulas de probabilidad de absorción en cada componente, de manera de construir una cadena auxiliar que conserve las mismas probabilidades de absorción que la cadena original. En la mayoría de los sistemas modelados, la cadena Y posee múltiples componentes conexas de tamaño reducido. Resolvemos entonces sistemas lineales de pequeño tamaño, y el tiempo de ejecución se reduce considerablemente.

Las clases utilizadas en la implementación del núcleo aparecen en la figura 5. Las clases *arrayOfStates*, *arrayOfStatesInMem* y *hashTable* son utilizadas para almacenar los estados del sistema teniendo en cuenta la eficiencia de acceso. La clase *sparseMatrix* sirve para almacenar matrices ralas, como es el caso de la matriz de transiciones M.

Las dos clases más importantes son *system* y *state*. Ambas son clases virtuales, que no pueden ser utilizadas de forma directa (no se puede definir directamente objetos de estas clases). La clase *system* describe los datos y el comportamiento genérico de todo sistema pausible de ser descrito con esta herramienta. Allí se encuentran las estructuras de datos para almacenar el estado inicial, el espacio de estado, y la matriz de las transiciones. En su especificación de interface encontramos (entre otros) los métodos siguientes:

- *fillStateList* - método de construcción del espacio de estado $\mathcal{E}$;
- *canonical* - método de eliminación de los estados no tangibles;
- *transitions* - método que produce la lista de las transiciones posibles a partir de un estado.

Los dos primeros métodos están implementados en la definición de la clase *system*, no así el último, que es un método virtual[2] que será implementado junto con la definición de las clases derivadas de *system*.

La clase *state* posee solamente métodos virtuales, los más importantes de los cuales son:

- *f1hash*,*f2hash* - funciones de hash utilizadas para almacenar el espacio de estado;

[1] Notation: $\omega_{\star j}$ es el vector columna dado por ω_{rj}, $j \in \mathcal{V}$;
$(\omega_{\star j})_{j \in \mathcal{Z} \setminus \mathcal{V}}$ es la matriz cuyas columnas son los vectores $\omega_{\star j}$

[2] En C++, un método virtual de una clase X debe ser obligatoriamente redefinido en todas las clases herederas de X. Este mecanismo es utilizado para permitir que X y sus clases derivadas compartan la misma interface para una función que tiene distintas implementaciones para cada sub-clase.

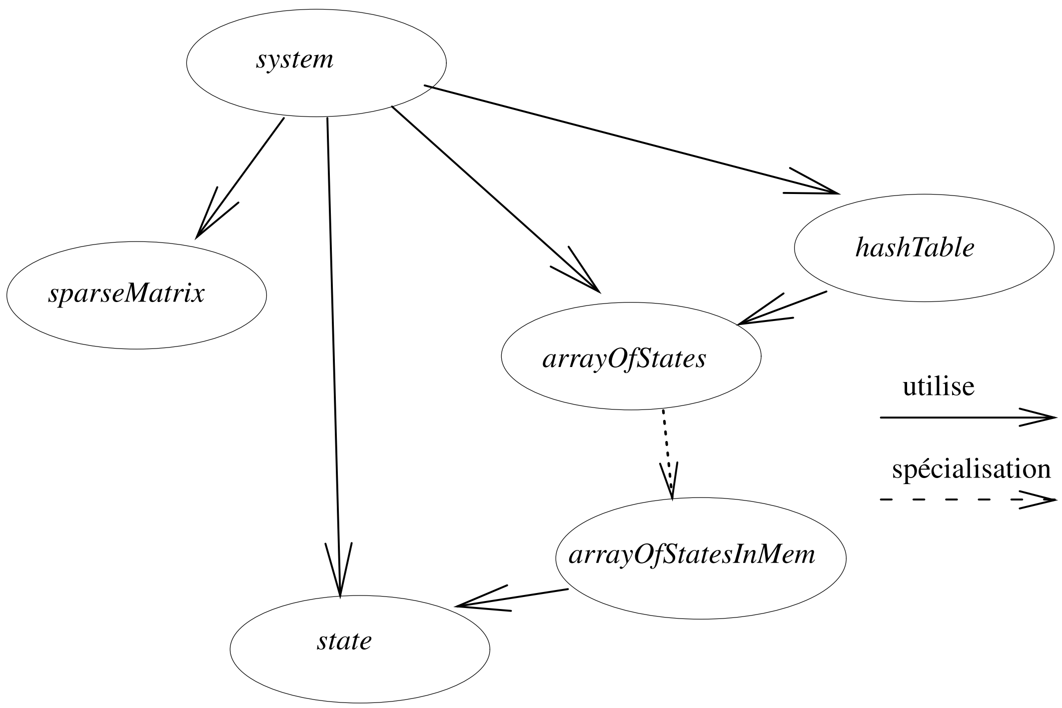

Figura 5 : Clases que componen el núcleo

- *isVanishing* - método que proporciona el tipo (tangible o no tangible) de un estado;
- *perfLevel* - método que proporciona el nivel de recompensa asociado a un estado.

3.2 Intérprete para "bolas y urnas"

El segundo nivel de la biblioteca está integrado por las clases derivadas de *system* y *state* que integran las características específicas del formalismo de *bolas y urnas*. Las clases definidas son *bab*, heredera de la clase *state*, y la clase *babSystem*, heredera de la clase *system*. En la figura 6 se presentan las relaciones de jerarquía y de uso de estas clases.

La clase *babSystem* incorpora, además de las estructuras de datos heredadas de la clase *system*, otras que son necesarias para la representación de un modelo de bolas y urnas:

- el número de urnas del modelo;
- los números máximos de bolas contenidas en esas urnas;
- las transferencias posibles entre las urnas, almacenadas como una matriz rala.

Entre los métodos de esta clase encontramos:

- *babSystem* - el método de construción de los objetos de clase *babSystem*.

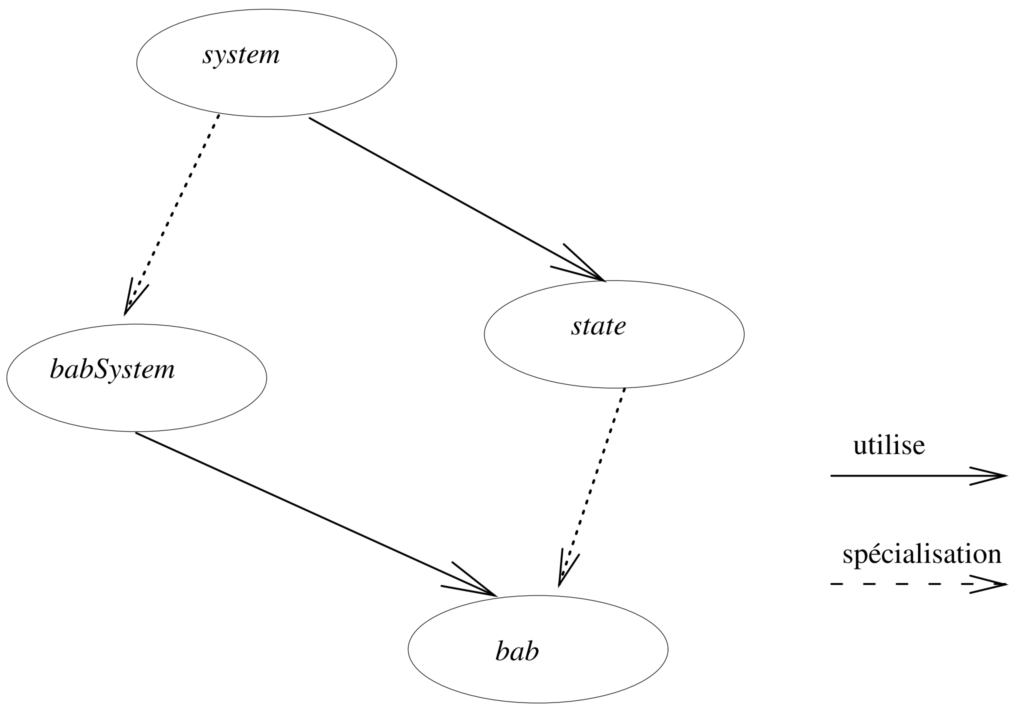

Figura 6 : Clases para el formalismo *bolas y urnas*

- *transitions* - este método, que da la lista de las transiciones posibles a partir de un estado, es parte integrante de la especificación de la clase *system*. Su implementación se realiza en *babSystem* a partir de los métodos *rate* y *vanishing*.

- *rate* - esta función proporciona la tasa de una transferencia de una urna hacia otra, cuando el sistema se encuentra en un estado dado.

- *instantaneous* - este método controla la existencia de modificaciones instantáneas a continuación de una transferencia, y proporciona el estado o la lista de estados que pueden ser alcanzados a partir de estas modificaciones.

Los dos últimos métodos no integran la biblioteca propiamente dicha, sino que deben ser proporcionados por el usuario como parte de la especificación del sistema modelado.

La clase *bab* contiene toda la información necesaria para almacenar y manipular un estado de un sistema descrito en términos de *bolas y urnas*. Esta clase incluye la implementación de métodos virtuales que habían sido definidos en la clase *state*, en particular las funciones de hash, y los operadores de comparación, que son utilizados como operaciones de base en los métodos de las clases *system* y *babSystem*. El método *perfLevel* que da el nivel de recompensa no está implementado en la biblioteca, dado que es parte de la especificación del sistema modelado, y debe ser entonces proporcionado por el usuario.

Para construir un modelo *bolas y urnas* es necesario llamar al método de creación proporcionando las estructuras de datos con los valores correspondientes al sistema modelado (el número de urnas, el máximo número de bolas en cada urna, el estado inicial del sistema y la matriz de transferencias entre urnas). Es necesario tambien suministrar los tres métodos *babSystem::instantaneous*, *babSystem::rate* y *bab::perfLevel* implementados en C++.

4 Expresión de otros formalismos en términos de "bolas y urnas"

En esta sección vemos métodos de pasaje de distintos formalismos al formalismo de *bolas y urnas*. Estos métodos han sido implementados sobre la biblioteca descrita en la sección precedente, con la finalidad de generar los procesos markovianos subyacentes.

El objetivo de esta parte de nuestro trabajo es demostrar la viabilidad del uso de un formalismo de nivel más alto que los procesos markovianos como un "lenguaje ensamblador" para la descripción de modelos de seguridad de funcionamiento. La incorporación de nuevos formalismos se realiza entonces a través de su descripción en términos del formalismo de base, dando al usuario una biblioteca de métodos de generación ya probada y de acceso transparente.

4.1 Redes de filas de espera

En una descripción *bolas y urnas* de una red de filas de espera podemos representar las estaciones por grupos de urnas y los clientes que circulan en la red por bolas en estas urnas. Las características de una estación serán modeladas por las transferencias en las urnas correspondientes. El ruteo de los clientes en la red se representa como transferencias entre las urnas asociadas a las distintas estaciones. Damos a guisa de ejemplo la descripción de algunos sub-sistemas importantes (tipos de estación), que suponemos formando parte de una red.

- Una estación con un solo servidor exponencial, disciplina FIFO:
  - una sola urna i que contiene el número de clientes en la fila;
  - transferencias de la urna i hacia la urna correspondiente a la fila de espera de la estación siguiente en la red . La tasa de la transferencia es igual a la tasa de servicio del servidor (ponderada eventualmente por la probabilidad p_j correspondiente).
- Una estación con k servidores exponenciales, disciplina FIFO:
  - una sola urna que contiene el número de clientes en la fila;
  - transferencias de la urna i hacia la urna correspondiente a la fila de espera de la estación siguiente en la red. La tasa de las transferencias es:
    $\min(k, \text{número de bolas en la urna}) \times (\text{tasa de servicio})$.
- Una estación con un servidor cuyo tiempo de servicio sigue una distribución de Cox de n fases:
  - dos urnas i y $i+1$, el número de bolas en la primera corresponde al número de clientes, y el número de bolas en la segunda corresponde a la fase del servicio;
  - transferencias de la urna i hacia la estación siguiente, con tasa correspondiente a la fase indicada en la urna $i+1$ multiplicada por la probabilidad de partida del servidor en esta fase; el contenido de la urna $i+1$ pasa a una sola bola para indicar el comienzo del servicio del próximo cliente;
  - transferencias de la urna $i+1$ hacia si misma (correspondiente al pasaje a la próxima fase del servicio) con tasa correspondiente a la fase indicada en la urna $i+1$ multiplicada por la probabilidad de permanencia. Esta transición aumenta en 1 el número de bolas en la urna $i+1$.

estas descripciones como bloques para construir el modelo de una red de filas de espera cualquiera.

La representación de subsistemas más generales es más o menos inmediata (modelos multiclases, ruteos en función del estado de la red, fenómenos de bloqueo, etc.).

4.2 Modelos MRM

Los modelos del tipo *"máquinas y reparadores"* (en inglés MRM – Machine Repairmen Model) son apropiados para sistemas integrados por un conjunto de máquinas, o más generalmente de componentes cualquiera, que pueden fallar, y un conjunto de estaciones de reparación de dichos componentes. Las máquinas pertenecen a K clases diferentes, cada una de las cuales posee un comportamiento característico (distribución del tiempo hasta una falla, distribución del tiempo de reparación). Hay m_i máquinas de clase i. Los reparadores pueden ser también de distintos tipos, y se caracterizan por la ley del tiempo de reparación y la disciplina de la fila de espera. Para que el sistema funcione correctamente debe haber por lo menos min_i máquinas de clase i en funcionamiento. El formalismo MRM puede ser enriquecido con dependencias entre los componentes y los reparadores (entre las más comunes está la propagación de fallas) y dependencias de funcionamiento o reparación (número mínimo de componentes en falla antes de comenzar la reparación, por ejemplo). Diversas herramientas emplean extensiones del formalismo MRM para describir modelos de seguridad de funcionamiento (ver [Car91], [GLT85], [Goy87], [BHL$^+$93], [NA80]).

Por ejemplo, podemos definir una familia de modelos MRM dada por el número K de tipos de máquinas diferentes, para cada tipo i la cantidad m_i de máquinas, las tasas de falla λ_i y de reparación μ_i, el número mínimo min_i de máquinas de cada tipo que deben estar en funcionamiento para que el sistema se encuentre en estado operativo, y la cantidad y las características de los reparadores.

Veamos una breve descripción de la especificación *bolas y urnas* de los modelos MRM siguientes:

- K tipos de máquinas diferentes, número infinito de reparadores, distribución del tiempo de falla y de reparación exponencial;
- K tipos de máquinas diferentes, un reparador por cada tipo de máquina, distribución del tiempo de falla y de reparación exponencial;
- K tipos de máquinas diferentes, un solo reparador, distribución del tiempo de falla y de reparación exponencial, prioridades de reparación (no preemptivas).

La representación elegida para el primer conjunto de modelos utiliza $2N$ urnas. En las urnas 1 a N se encuentran las bolas correspondientes a las máquinas en buen funcionamiento y en las urnas $N+1$ a $2N$ las bolas correspondiente a las máquinas en reparación. Las transferencias posibles son de dos tipos: aquellas correspondientes a la falla de una máquina de tipo i se realizan de una urna i ($i \leq N$) hacia una urna $N+i$ con tasa $n_i\lambda_i$, donde n_i es la cantidad de bolas en la urna i. Aquellas correspondientes a una reparación van de una urna $N+i$ hacia una urna i con tasa $n_{N+i}\mu_i$.

Para el segundo conjunto de modelos la representación *bolas y urnas* es casi idéntica: la única diferencia es que las transferencias de una urna $N+i$ hacia una urna i se realizan a un tasa μ_i en lugar de $n_{N+i}\mu_i$ (debido a la existencia de un único reparador por tipo de máquina).

Para representar la última clase de modelos tomamos $2N+1$ urnas. Las primeras $2N$ urnas tienen la misma función que en los casos precedentes; la última urna memoriza cual es el tipo de máquina que está siendo reparada actualmente (hay un solo reparador para todas las máquinas). Las transiciones de una urna i ($i \leq N$) hacia una urna $N+i$ son de tasa $n_i\lambda_i$. Tenemos una sola transferencia ligada a las reparaciones que se efectúa de la urna $2N+1$ hacia ella misma y es seguida de una transición instantánea que suprime una bola en la urna $N + n_{2N+1}$, agrega una bola en la urna n_{2N+1} y busca en las urnas $N + 1$ a $2N$ según el orden de prioridades de las máquinas para encontrar la proxima máquina a reparar (este será el nuevo valor en la urna $2N + 1$; si no hay ninguna máquina para reparar el valor a almacenar será 0).

4.3 Redes de Petri estocásticas

Las redes de Petri estocásticas ([Nat80], [Mol81]) son una extensión del formalismo bien conocido de redes de Petri. Conservan las propiedades estructurales de estas últimas pero asocian a cada transición una variable aleatoria que corresponde al tiempo necesario para franquearla efectivamente, una vez que ha sido activada. Distintas variantes de las redes de Petri estocásticas han sido utilizadas en herramientas de cálculo de la seguridad de funcionamiento. Entre ellas podemos citar las redes estocásticas de actívides (SAN – Stochastic Activity Networks, ver [SM86], [SO93]), las redes de Petri estocásticas extendidas (ESPN, ver [DTGN84]), las redes de Petri estocásticas generalizadas (GSPN, ver [AMBC88], [BAA+93], [Chi91]).

Un modelo de las redes de Petri estocásticas generalizadas (incluyendo arcos inhibidores) se define por $(P, T, A, \mathrm{Pre}, \mathrm{Post}, M_0, L)$, siendo:

- $P = \{p_1, \cdots, p_n\}$ el conjunto de lugares de la red;
- $T = \{t_1, \cdots, t_m\}$ el conjunto de las transiciones;
- $A = (P \times T) \cup (T \times P)$ el conjunto de los arcos;
- $\mathrm{Pre}(p_i, t_j) \in \{1..\infty\} \cup \{-1\}$ la ponderación del arco de entrada (p_i, t_j) (-1 si es un arco inhibidor);
- $\mathrm{Post}(t_j, p_k) \in \{1..\infty\}$ la ponderación del arco de salida (t_j, p_k);
- $M_0(p_i)$ el marcado inicial del lugar p_i;
- $L(t_i)$ la tasa de la transición t_i ($L(t_i) \in \overline{\mathbb{R}}^+, \forall t_i$).

Los métodos implementados permiten tomar como entrada un archivo con la descripción de la red y crean una descripción *bolas y urnas* equivalente que es utilizada para realizar la generación del proceso de Markov subyacente.

Esta descripción utiliza $(n + m)$ urnas; las urnas 1 a n corresponden a los lugares de la red de Petri y las bolas en dichas urnas corresponden a las fichas en los lugares de la red. Las urnas $n + 1$ a $n + m$ corresponden a las transiciones: pueden estar vacías (transición no franqueable) o contener una bola (transición franqueable).

Las únicas transferencias posibles son de las urnas $n + j$ hacia ellas mismas. Estas transferencias tienen los siguientes efectos:

1) modificar el marcado de la red, es decir el contenido de las urnas 1 a n, siguiendo las pre y post condiciones de la red de Petri;

2) como consecuencia de la modificación del marcado de la red, el estado (franqueable o no franqueable) de las transiciones puede alterarse. En este caso es necesario modificar el contenido de las urnas correspondientes a las transiciones que cambiaron de estado. La primer opción consiste en verificar el estado de todas las transiciones. Una alternativa más eficiente recalcula solo el estado de aquellas transiciones que poseen un lugar de entrada en común con los lugares que la transición t_j modificó. En la implementación efectuada utilizamos una matriz *affects[i,j]* para almacenar esta información.

El efecto de una transferencia de una urna $n + j$ hacia ella misma se calcula entonces con el siguiente algoritmo:

```
para todos los arcos de entrada no inhibidores (p_i, t_j) de la transición t_j
        quitar Pre(i,j) bolas de la urna i
para todos los arcos de salida (t_j, p_k) de la transición t_j
        colocar Post(j,k) bolas en la urna k
para todas las transiciones l
        si  affects[j,l]
                controlar la situación de la transición t_l
                si t_l es franqueable
                        n_{n+l} := 1
                sino
                        n_{n+l} := 0
                fin si
        fin si
```

Una transición t_l del red de Petri es franqueable cuando todas sus precondiciones son satisfechas por el marcado:

```
franqueable=TRUE
para todos los lugares i tales que existe un arco de entrada (p_i, t_l)
        si Pre(i,l)=-1 (arco inhibidor)
                franqueable=franqueable AND (n_i = 0)
        sino
                franqueable=franqueable AND (n_i >=Pre(i,l))
        fin si
```

5 Ejemplo: una familia de arquitecturas tolerantes a fallos

Esta familia de modelos corresponde a un tipo de arquitectura multiprocesador de memoria compartida, tolerante a fallos, desarrollada en el marco de un proyecto europeo ESPRIT[3]. El sistema posee P procesadores independientes sujetos a fallos con período de funcionamiento de distribución exponencial (de tasa λ). Cuando un procesador falla el sistema efectúa un procedimiento de recubrimiento del fallo que ocupa un tiempo exponencial (tasa μ) y que termina con probabilidad c en una recuperación del procesador o con probabilidad $1 - c$ en la desactivación del procesador dañado y una reconfiguracion degradada del sistema con los procesadores restantes. Si no queda ningún procesador en buen estado, el sistema completo está en falla y se detiene completamente para sufrir una intervención externa (la reparación externa no se considera como parte integrante del sistema). Existen otros L elementos de arquitectura (por ejemplo memorias, buses de entrada/salida, controladores de disco, etc.) duplicados para aumentar la confiabilidad. Todos estos componentes están sujetos a fallas con tasa diferentes para cada tipo. Cuando un componente de un tipo determinado falla el sistema continúa su funcionamiento con el otro componente del mismo tipo. Si a su vez el segundo componente falla el sistema se detiene para recibir un mantenimiento externo (sistema en estado de falla). En la figura 7 se presenta un esquema general de la arquitectura.

[3] proyecto Esprit II Fasst: *Fault Tolerant Architectures with Stable Storage Technology*

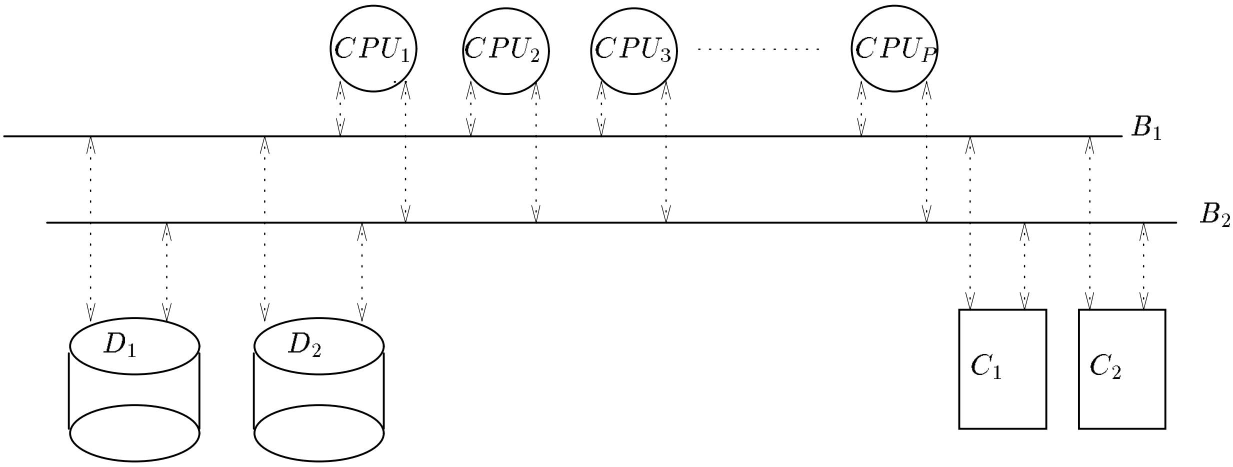

Figura 7 : Ejemplo de arquitectura FASST

El código implementado toma como entrada los siguientes parámetros, que describen completamente un modelo: el número P de procesadores, la tasa λ de fallo individual de los procesadores, la tasa μ del tiempo de recubrimiento (puede ser infinita, lo que equivale a un tiempo de recubrimiento nulo), el coeficiente de recubrimiento c, la cantidad L de otros elementos duplicados en la arquitectura y la tasa de falla individual λ_i de cada tipo i de componente duplicado.

El modelo *bolas y urnas* asociado tiene $L + 2$ urnas. Las primeras L urnas corresponden a los L componentes duplicados de la arquitectura; pueden contener 2 bolas (las dos unidades de este tipo de componente funcionan), 1 (una sola unidad funciona), o 0 (ninguna unidad funciona, lo que implica que el sistema está en falla). La urna $L + 1$ posee una cantidad de bolas igual al número de procesadores en funcionamiento y la urna $L + 2$ es un indicador del estado del sistema: si no contiene ninguna bola, el sistema está en funcionamiento; si contiene una bola, se ha producido un fallo en los procesadores y el sistema está efectuando el recubrimiento. El estado de falla del sistema corresponde al estado con todas las urnas vacías. La evolución del sistema está determinada por los eventos descritos a continuación.

- La falla de un componente de tipo i ($1 \leq i \leq L$) se modela por una transferencia de la urna i hacia ella misma con tasa $n_i\lambda_i$, siendo n_i el número de bolas en la urna, seguida de una modificación instantánea que disminuye en 1 el número de bolas en esta urna (cuando la urna está vacía se debe vaciar también todas las otras urnas, lo que corresponde al estado de falla del sistema).

- El fallo de un procesador corresponde a una transferencia de la urna $L + 1$ hacia la urna $L + 2$ con tasa $n_{L+1}\lambda$, seguido por una modificación instantánea. El efecto combinado de estas dos acciones es mantener incambiado el número de bolas en $L + 1$, y agregar una bola en $L + 2$. Esta transición es posible solo cuando la urna $L + 2$ está vacía.

- El procedimiento de recubrimiento se modela mediante una transferencia de la urna $L + 2$ hacia la urna $L + 1$ con tasa μ, seguida por una de las transiciones instantáneas, elegidas con probabilidades c y $1 - c$ respectivamente. En la primera, que corresponde a un fallo transitorio del procesador, la urna $L+1$ permanece con el mismo número de bolas. La alternativa corresponde a la eliminación del procesador fallado y a la reconfiguración del sistema; por lo tanto, se disminuye en 1 el número de bolas de la urna $L+1$. En los dos casos la urna $L+2$ continúa vacía. Cuando luego de la eliminación de un procesador la urna $L+1$ no posee ninguna bola se ha llegado al estado de falla del sistema y todas las urnas deben ser vaciadas (estado absorbente).

Se considera que mientras dura el procedimiento de recubrimiento, el sistema no está operativo. Para esta familia de modelos el número de estados del proceso markoviano subyacente es igual a $2^{L+1}P+1$.

6 Conclusiones

Hemos explorado en este trabajo el uso del formalismo de *bolas y urnas* como un "lenguaje ensamblador" para la descripción de modelos de seguridad de funcionamiento. La elección de un langage orientado a objetos (C++) para la implementación realizada permitió realizar una arquitectura en capas sumamente flexible. El usuario puede definir clases adaptadas a otros tipos de modelos, aprovechando el núcleo de generación de procesos markovianos subyacente. La incorporación de nuevos formalismos puede también realizarse a través de su descripción en términos del formalismo de base, dando al usuario una biblioteca de métodos ya probada, y de acceso transparente.

Un problema no resuelto es el del crecimiento exponencial del tamaño del espacio de estado. Para sistemas muy grandes una solution posible consiste en implementar técnicas de truncación del espacio de estado en el primer nivel de la biblioteca, de forma transparente al usuario. Un enfoque alternativo consiste en emplear simulación, que no requiere la generación de todo el espacio de estado. Actualmente estamos estudiando la incorporación de primitivas de simulación a nivel del núcleo. También consideramos la posibilidad de incorporar métodos de cálculo en el núcleo, capaces de ejecutarse a medida que se realiza la generación.

Otra perspectiva consiste en generalizar la clase de modelos que la biblioteca es capaz de tratar. De particular interés son la inclusión de cadenas de Markov de tiempo discreto, la incorporación de distribuciones más generales (Erlang, PH) en los tiempos de transición y el estudio de procesos semi-markovianos. Igualmente importante resulta agregar métodos jerárquicos de modelado y presentación de resultados a todos los niveles.

Bibliografía

[AMBC88] M. Ajmone Marsan, G. Balbo, and G. Conte. *Performance Models of Multiprocessor Systems*. The MIT Press, 1988.

[BAA+93] C. Béounes, M. Aguéra, J. Arlat, S. Bachmann, C. Bourdeau, J.E. Doucet, K. Kanoun, J.-C. Laprie, S. Metge, J. Moreira de Souza, D. Powell, and P. Spiesser. SURF-2: A program for dependability evaluation of complex hardware and software systems. In *FTCS-23 Digest of Papers*, pages 668–673, Toulouse, France, June 1993. IEEE Computer Society Press.

[BHL+93] Alvin M. Blum, Philip Heidelberger, Stephen S. Lavenberg, Marvin K. Nakayama, and Perwez Shahabuddin. System availability estimator (save) language reference and user's manual version 4.0. Research Report RA 219 S (82923), T. J. Watson Research Center - IBM, Yorktown Hts, NY 10598, June 1993.

[Car91] J. A. Carrasco. Hierarchical, object oriented modeling of fault-tolerant computer systems. In *IEEE CompEuro 91 Conference*, pages 452–456, Bologne, 1991. IEEE Computer Society Press.

[Chi91] Giovanni Chiola. GreatSPN 1.5 software architecture. In *5th. Modelling Techniques and Tools for Computer Performance Evaluation*, pages 117–132, Torino, Italy, February 1991.

[Cia93] G. Ciardo. Generalized and deterministic stochastic petri nets (GSPN and DSPN). In *Pre-Workshop Tutorials of the 5th. International Workshop on Petri Nets and Performance Models (PNPM93)*, Toulouse, France, October 1993.

[DTGN84] J.B. Dugan, K.S. Trivedi, R.M. Geist, and V.F. Nicola. Extended stochastic Petri nets: applications and analysis. In *Performance'84*, pages 507–519, 1984.

[GLT85] Ambuj Goyal, Stephen S. Lavenberg, and Kishor S. Trivedi. Probabilistic modeling of computer system availability. Research Report RC 11076, Thomas J. Watson Research Center - IBM, Yorktown Hts, NY 10598, March 1985.

[Goy87] Ambuj Goyal. System availability estimator (save) user's manual version 2.0. Research Report RC 12517, Thomas J. Watson Research Center - IBM, Yorktown Hts, NY 10598, February 1987.

[Lap92] Jean-Claude Laprie, editor. *Dependability: Basic Concepts and Terminology in English, French, German, Italian and Japanese*. Springer-Verlag, 1992.

[Mey80] J.F. Meyer. On evaluating the performability of degradable computing systems. *IEEE Transactions on Computers*, C-29:720–731, 1980.

[Mol81] M.K. Molloy. *On the Integration of Delay and Throughput Measures in Distributed Processing Models*. PhD thesis, University of California, Los Angeles, 1981.

[NA80] Y.W. Ng and A.A. Avižienis. A unified reliability model for fault-tolerant computers. *IEEE Transactions on Computers*, C-29(11), November 1980.

[Nat80] S. Natkin. *Réseaux de Petri Stochastiques*. PhD thesis, CNAM-Paris, 1980.

[SM86] W.H. Sanders and J.F. Meyer. METASAN: A performability evaluation tool based on stochastic activity networks. In *Proceedings of the Fall Joint Computer Conference*, pages 807–816, Dallas, November 1986. ACM / Computer Society of IEEE, IEEE Computer Society Press.

[SO93] William H. Sanders and W. Douglas Obal, II. Dependability evaluation using UltraSAN. In *FTCS-23 Digest of Papers*, pages 674–679, Toulouse, France, June 1993. IEEE Computer Society Press.

[Ste91] William J. Stewart. MARCA: Markov chain analyzer, a software package for Markov modeling. In William J. Stewart, editor, *Numerical Solution of Markov Chains*, volume 8 of *Probability: Pure and Applied*, chapter 3, pages 37–61. Marcel Dekker, Inc, 1991.