Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

Animación de Algoritmos Distribuidos para Detección de Terminación

Trabajo Especial de Grado presentado ante la ilustre
Universidad Central de Venezuela por el Bachiller:

Lam Chang, Johnny Mintak

Para optar al título de Licenciado en Computación

Tutor:
Profesora Claudia León

Caracas 29 de Octubre de 2009

ACTA

Quienes suscriben, miembros del Jurado designado por el Consejo de Escuela de Computación, para examinar el Trabajo Especial de Grado presentado por el Bachiller Johnny Mintak Lam Chang, C.I. 14283557, con el título “Animación de Algoritmos Distribuidos para Detección de Terminación”, a los fines de optar al título de Licenciado en Computación, dejen constancia de lo siguiente:

Leído como fue, dicho trabajo por cada uno de los miembros del jurado, se fijó el día 29 de Octubre de 2009 a las 10:00 a.m., para que sus autores lo defendieran en forma pública, lo que hicieron en la Sala 1 de la Escuela de Computación, mediante una presentación oral de su contenido, luego de lo cual respondieron las preguntas formuladas. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo.

En fe de lo cual se levanta la presente Acta, en Caracas los 29 días del mes de Octubre del año dos mil nueve dejándose también constancia de que actuó como Coordinador del Jurado la profesora tutor Claudia León.

_____ _____

Prof. Claudia León (Tutor) Prof. Andrés Sanoja (Jurado)

_____ _____

Prof. Nora Montaño (Jurado) Prof. Guy Vernáez (Jurado suplente)

_____

Prof. Robinson Rivas (Jurado suplente)

RESUMEN

Se desarrolla una herramienta monoprocesadora que anima modelos de ejecución de los sistemas distribuidos con el fin de soportar el proceso didáctico en la enseñanza de los algoritmos distribuidos. Las animaciones desarrolladas se centran en la detección de terminación de los procesos distribuidos, permitiéndose la parametrización de variables como el volumen de actividades, la productividad y el grafo de comunicación del sistema. Los métodos de detección han sido clasificados en tres grupos respondiendo a ciertas igualdades fundamentales de la gran cantidad de propuestas que existen para el problema de terminación. Adicionalmente se realiza una comparación de las distintas implementaciones de los métodos hechos, en términos de los datos almacenados y mensajes utilizados. Con esto se crea un modelo básico que sirve como prototipo para el desarrollo posterior de los sistemas estudiados, o bien sistemas simuladores de éstos.

Palabras claves: Algoritmos distribuidos, detección de terminación, animación multihilo.

ÍNDICE

INTRODUCCIÓN

En el campo de la ciencia una significante porción de las postulaciones derivadas de investigaciones se deben a experimentos y trabajos prácticos. A pesar que los algoritmos distribuidos son modelos computacionales bastantes maduros, la correspondencia entre el desarrollo e implementación de los diseños y esquemas con la parte conceptual no es exacta. Los modelos abordan aspectos fundamentales de los sistemas, como la asincronía, el tráfico, el conocimiento local, las fallas, etcétera. El perfeccionamiento de dichos modelos puede lograrse con una sostenida base matemática y una fuerte experimentación. Las simulaciones ahorran tiempo y costos para lograr tales metas, y son de tal importancia para la ciencia que existen estudios numéricos de las aproximaciones de las simulaciones a la realidad.

Una gran porción del estudio de los sistemas distribuidos está orientada a la detección de terminación. En primer lugar esto se debe a la gran variedad de modelos que pueden surgir debido a las características de los sistemas. Segundo, se debe a que el problema de detección de terminación, siendo un caso tan fácil de entender pero su solución no tan trivial, es un buen candidato para ilustrar las circunstancias de un ambiente distribuido. Tercero, se ha observado que las dificultades fundamentales del problema de detección de terminación están intimamente asociados a los algoritmos de detección de estados globales y los algoritmos de detección de abrazo mortal [MT91].

En este trabajo describimos animación de varios métodos de detección de terminación. Uno está basado en un ciclo de consultas y está diseñado para abordar directamente el problema de nodos procesadores que se reactivan con mensajes de aplicación. Otro está basado en la distribución y

recuperación de valores. El último método aprovecha la forma de trabajo de los sistemas de computación.

La utilización de analogías visuales de datos, modelos o conceptos es un recurso que explota la mayor capacidad de reconocimiento visual del ser humano, y al brindar un soporte computacional más dinámico e interactivo, se promueve mejor asimilación al momento de comprender un concepto o interpretar una estructura de datos. Aquí proponemos aprovechar el software para ofrecer la posibilidad de una enseñanza no lineal en la que no es preciso seguir una determinada trayectoria, sino más adaptada a la cotidianidad y a la interactividad como medio de captación y entendimiento.

El apartado más importante de este documento es el Capítulo 2, donde se presenta la estructura fundacional sobre la que se erige el Trabajo Especial de Grado, lo concerniente a su desarrollo y los análisis conclutorios; como sección introductoria está el Capítulos 1, el cual esboza en tres categorías los algoritmos de detección de terminación con finalidad de repaso de los métodos o bien para el lector precoz. El complemento del Capítulo 2 está en los Anexos, con los datos de las pruebas realizadas al software durante y después de su desarrollo.

CAPÍTULO 1: ALGORITMOS DE DETECCIÓN DE TERMINACIÓN

Existe una amplia gama de algoritmos de detección de terminación las cuales incurren en técnicas regularmente similares y en las que a veces, aún pertenecientes a estudios y autores distintos, tienen en esencia las mismas características lógicas para detectar la terminación. Aún así, es difícil clasificar a cada uno en apenas uno de los tres siguientes grupos, bien porque tiene propiedades de más de un grupo o porque tiene características muy particulares y tal vez merece una distinción separada. Sin embargo, por facilitar a la claridad resaltamos tres tipos de técnicas.

1.1.- MÉTODOS DE PROPAGACIÓN DE CONSULTA [DFvG83] [Mat87] [Mat90] [Top84]

En estos métodos un proceso coordinador inicia la detección de terminación a través de mensajes especiales, los cuales al recorrer completamente el sistema, se determina si todos los procesadores están en posición para terminar. Se trata de recorrer completamente el sistema para recolectar información uno por uno en los nodos, por lo que la consulta iniciada por el coordinador se propaga como una especie de onda controlada. Cuando el proceso coordinador termina todas sus actividades, inicia la actividad de detección liberando al sistema el mensaje especial. Si el procesador que recibe el mensaje ha culminado todas sus actividades y espera la terminación, reenvía el mensaje especial al siguiente vecino. Si no ha culminado, espera su propia culminación y luego emite el mensaje especial. El mensaje especial recorre completamente así el sistema hasta llegar nuevamente al proceso originador. Cuando el coordinador descubre que todos los procesadores han culminado sus actividades, libera un mensaje notificador para informar a todos los procesadores que el sistema puede satisfactoriamente finalizar.

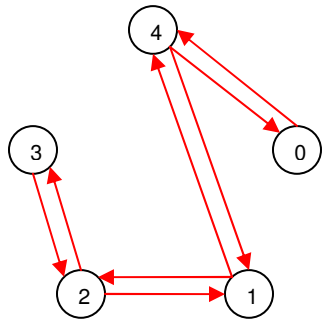


Fig. 1.- La consulta se propaga y regresa a través de mensajes especiales en todo el sistema.

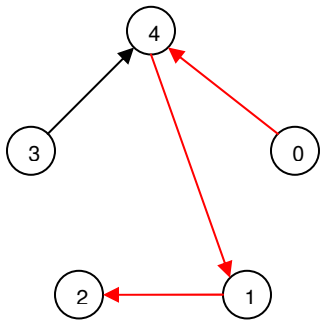


Fig. 2.- Antes de finalizar la consulta un mensaje puede despertar un nodo ya consultado.

Un procesador reenvía el mensaje especial de detección cuando ha culminado sus procesos, pero luego de este

momento, pueden ocurrir dos situaciones: Que reciba el mensaje de finalización general del sistema, por lo que termina su espera y ejecución; o que reciba un mensaje de aplicación de algún proceso foráneo, lo que puede significar reanudar la ejecución o disparar procesos locales consecuentes, en dicho caso, el sistema no necesariamente puede finalizar. Esta última situación es el caso borde inherente más claro del algoritmo de detección. Existen problemas generales como la ruptura de la topología o la pérdida del mensaje especial, pero son problemas propios de los sistemas de computación.

Dado que un procesador que ha notificado su culminación y espera la finalización general puede reactivarse, el regreso del mensaje especial al proceso originador sólo garantiza que al menos alguna vez todos los procesos foráneos y locales finalizaron sus actividades. El problema no radica en que un procesador salga de su estado pasivo y vuelva a trabajar, porque bien podría llevar a término todas sus nuevos procesos y volver a estado pasivo justo antes de decretar la terminación del sistema y no alterar la información que se ha recolectado hasta ese momento o la información que se va a recolectar. O bien, su reactivación no tiene que ver con la información a procesar. Dado el problema obvio existente, la opción es notificar al proceso coordinador que la finalización general no puede aún darse y debe realizarse otro ciclo de consulta.

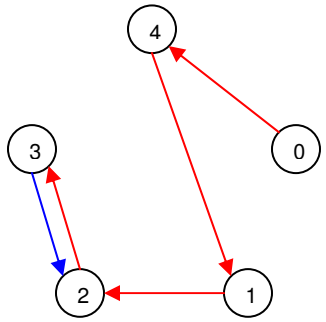


Fig. 3.- Un nodo que haya mandado mensajes ordinarios altera el mensaje especial.

La finalización general entonces se decreta posiblemente después de dos o más consultas, vueltas del mensaje especial por toda la topología. Efectivamente, si en el sistema hubo al menos un mensaje de aplicación, se requiere un segundo ciclo de consulta. Como el problema se origina con los mensajes en tránsito destinado a los procesadores ya pasivos, la solución es que un procesador pasivo reenvíe el mensaje especial cuando sea su turno pero lo altere si ha enviado mensajes de aplicación desde la última consulta. Cuando el proceso coordinador recibe el mensaje especial alterado de vuelta, genera otro ciclo de consulta liberando nuevamente otro mensaje especial al sistema. Así, el decreto de terminación final garantiza que no hubo mensajes en tránsito entre procesos remotos.

Existen distintas variantes del método concernientes a la topología a la que quiera aplicarse. Por ejemplo, **[DFvG83]** formaliza el método para una topología de anillo, topología en la que muy didácticamente se puede visualizar el problema inherente a la reactivación de procesadores ya

pasivos. **[Mat90]** explica la técnica desde un punto de vista más general pero informalmente, donde los eventos que cuentan no son los de envío de mensaje sino de recepción. De igual manera en **[Mat87]** donde previamente esboza un algoritmo parecido con ondas de consulta. En **[Top84]** puede verse la explicación de un algoritmo similar para un árbol de expansión.

Formalizando el algoritmo:

Para todos los nodos procesadores se cumple que

```
if (I send an application message)
      Set myFlag;                // myFlag := 1
```

Adicionalmente para los nodos no coordinadores:

```
if (I receive the special message){
      while (I am Active){
            Keep the special message;
      }
      if (myFlag is Set){              // ¿myFlag == 1?
            Set special message;       // special message flag := 1
            Reset myFlag;              // myFlag := 0
      }
      Send (Special Message) to a neighbor process;
}
```

Adicionalmente para el nodo coordinador:

```
while (I am Active){
      Keep the special message;
}
Send (Special Message) to a neighbor process;


if (I receive the special message)
```

```
if (special message is Set) or (myFlag is Set){
      Reset special message;          // special message flag := 0
      Reset myFlag;              // myFlag := 0
      Send (Special Message) to a neighbor process;
}
else
      Global Quiescence;    // Algorithm ends.
```

1.2.- MÉTODOS BASADOS EN LA DISTRIBUCIÓN Y RECUPERACIÓN DE CRÉDITOS [Mat89] [DDE04]

Este algoritmo de detección es bastante versátil, dada su simplicidad lógica, su independencia de líneas de comunicación FIFO, la falta de necesidad de conocer la topología completa de red y la posibilidad de amoldarse a la iniciación asíncrona de procesos nuevos, es uno de los métodos de detección más conocidos. La técnica se basa en la distribución de una cantidad fija de créditos durante la vida del sistema. Al inicio, un nodo de procesamiento distribuye créditos a todos los nodos participantes del sistema distribuido. Mientras los créditos del sistema están en circulación, el sistema no ha finalizado. La cantidad de créditos en el sistema es fija, distribuidos no uniformente entre los procesos activos, los mensajes en tránsito y el procesador detector de la finalización, el cual determina la finalización general cuando los créditos dejan de circular y están todos reunidos nuevamente.

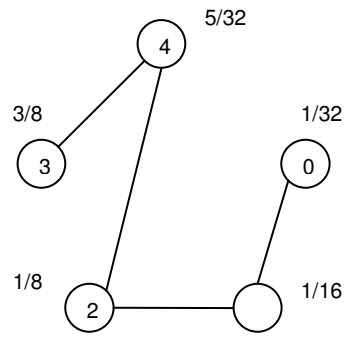


Fig. 4.- Una distribución de créditos entre procesos activos. Una cuarta parte del total de créditos se encuentra en mensajes en tránsito.

La cantidad total de créditos no es importante, tampoco es necesario que cada nodo tenga la misma cantidad mientras esté activo, ni las unidades que posee cada mensaje en tránsito. Mientras un nodo esté activo debe tener créditos, por eso el procesador inicial debe distribuir créditos a todo proceso que vaya a iniciar, lo cual constituye la primera premisa en la programación del método, debe conocerse la identificación y el número de procesos que va a iniciar al primer momento de distribuir los créditos o bien debe haber un proceso distribuidor de créditos a quien solicitar créditos para poder trabajar. Como la cantidad de créditos en cada punto no es importante, otorga la

flexibilidad de que nuevos procesos sean incorporados al sistema en el tiempo. Cuando los nodos finalizan sus procesos ordinarios o asignaciones de aplicación devuelven los créditos que retienen a un proceso acumulador de créditos, el cual puede ser el mismo distribuidor y que detectará la finalización de todos los procesos.

Como la dificultad en la detección de terminación está en los procesadores inactivos que salen de la pasividad y vuelven a trabajar, un proceso que mande un mensaje de aplicación a otro proceso foráneo, lo cual puede ser causante de nuevas actividades importantes en el proceso receptor, debe otorgarle parte de sus créditos en un mensaje de control. Si el nodo receptor reanuda sus actividades, y que consecuentemente podría afectar los resultados finales de la sesión distribuida, mantiene los créditos; por el contrario, si no se reactiva o su nuevo trabajo no altera los resultados del sistema, otorga los créditos recibidos al proceso detector. Así, mientras existan procesos activos o mensajes en tránsito, los créditos jamás se juntarán completamente en el procesador detector.

Dado que la detección de terminación se basa en la acumulación de los créditos totales, la topología de red o la forma de distribución de los créditos no afecta en lo absoluto el resultado final o el desempeño del algoritmo. La red distribuida puede ser muy extensa, los créditos pueden distribuirse a lo largo de ella sin que el proceso iniciador o el proceso detector de terminación sepan cuánto se han dividido los créditos. A menos que haya corrupción en los datos transmitidos, la detección no genera falsos positivos.

Por su simplicidad, el algoritmo se presta para diferentes configuraciones u optimizaciones. Para empezar, lo más intuitivo es pensar que cada vez que un proceso envía un mensaje a otro proceso foráneo, divide sus créditos a la mitad y le envía la mitad al otro proceso, además de ser una política didáctica, es lo ideal, pero existe la limitante que es la representación de los números fraccionarios en el computador, pero no necesariamente tiene que ser así. Los créditos transmitidos no tienen que ser fracciones, pueden ser número enteros, y como se mencionó antes, la forma de distribución de créditos no es significativa.

Otro punto, cuyos detalles técnicos escapan un poco a la esencia del algoritmo, es la forma de transmisión de los créditos. Pueden utilizarse mensajes de control adicionalmente a los mensajes de aplicación, el problema es cuando hay mucho retardo en el tráfico de mensajes. Aparentemente esto no afecta el desempeño del método, dado que los créditos en tránsito son créditos que no retiene el proceso detector. Pero supongamos el siguiente escenario: Un procesador P_1 ha culminado sus

procesos de aplicación, por lo que devuelve su créditos al sistema y queda sin créditos. En un tiempo siguiente *t*, el procesador P_2 le envía un mensaje de aplicación a P_1 y éste dispara un proceso ordinario. Como no puede ser simultáneo, en un tiempo siguiente a *t*, pero inmediato, P_2 también le envía el mensaje de control correspondiente con los créditos respectivos. Hasta este momento, todo marcha bien. La situación hipotética es que en ese instante hay retardo en la red. El mensaje de aplicación llega a P_1 en el tiempo t_i, el mensaje de control con los créditos no llega inmediatamente y por el retardo llega en el tiempo t_{i+j}. P_1 trabaja lo suficientemente rápido, termina su nueva asignación en el tiempo t_{i+x}, donde x<j, y como ha finalizado, mecánicamente entrega sus créditos al proceso detector, le envía cero (0) créditos. En el tiempo t_{i+j}, P_1 recibe los créditos que le correspondían pero ya no hace nada con ellos. El proceso detector no recibe los créditos por la finalización de P_1, y efectivamente no decreta la finalización general del sistema hasta recibir todos los créditos faltantes. En un tiempo futuro t_{i+y} todos los procesos terminan y devuelven sus créditos, P_1 nunca más es despertado y retiene los créditos que posee. Como consecuencia la terminación general no se detecta y nunca se anuncia. Esto quizás no afecta el desempeño del sistema ni altera los resultados obtenidos por el trabajo de los procesos, pero el método falla en este caso. La opción ideal es que los créditos a entregar viajen conjuntamente con los mensajes de aplicación, aunque los algoritmos de aplicación deben entonces rediseñarse para que forjen los mensajes con créditos. Otra solución es enviar los mensajes de control antes que los mensajes de aplicación, así, los procesos esperan la llegada de mensajes al recibir créditos. Esto hace la programación del algoritmo un poco más compleja al incurrir en el uso de colas, ya que a cada mensaje de control le corresponde un mensaje de aplicación y éstos deben estar identificados conjuntamente para evitar que los retardos y las llegadas fuera de orden induzcan a una situación similar a la expuesta y haya pérdida de créditos.

Como se dijo antes, la forma de distribución de créditos no es significativa, es decir, el método de acumulación y el cálculo de la división de créditos no es parte de la esencia básica del algoritmo. Pero sí es un aspecto interesante a tomar en cuenta. La totalidad de créditos en el sistema puede ser un número entero cualquiera, o bien un valor genérico que representa el 100% de créditos o unidades a distribuir. En computación, así como en las matemáticas, el número real 1.0 representa muchas veces el 100% de las cosas a dividir dado la versatilidad que tiene dicho número en ciertos cálculos. Esto no es tan importante en la programación. Dada la capacidad numérica de los computadores cualquier número es divisible y representable de alguna forma digitalmente. El problema electrónico con las matemáticas es el redondeo y la acumulación de errores, si bien la capacidad actual de los ordenadores es inmensa, ciertas previsiones deben tomarse. Otro asunto es

que la división de números muy grandes requiere mayor tiempo de procesamiento. Para todo esto, con respecto al algoritmo de detección, ciertas optimizaciones se proponen.

Trabajar con números muy grandes conlleva no sólo a mayor uso del procesador, también la posibilidad de error a causa de aproximaciones numérica aumenta. Como solución al problema anterior, se podría utilizar en vez de números fraccionarios, utilizar el logaritmo negativo binario y trabajar con matemática de números enteros. La cantidad real de créditos que tendría un proceso es 2^{-CR}, donde *CR* es la variable de crédito. Un proceso con ¼ de la totalidad de créditos del sistema tendría 2 unidades créditos, ya que $2^{-2} = 0.25 = ¼$ de créditos. El sistema binario es muy conveniente en este caso, nótese que 2^{i+1} es la mitad de 2^{i}. Entonces, cuando un proceso envía un mensaje de aplicación y otorga sus créditos, primero incrementa *CR* en una unidad y luego envía *CR* créditos, `CR:=CR+1, SEND(CR)`. Así, reduce sus créditos reales a la mitad y entrega la mitad. Si un proceso recibe un mensaje y está en modo pasivo, recibe los créditos transmitidos $CR:=CR_{received}$. Los procesos no necesitan calcular cuántos créditos reales tienen con el logaritmo negativo binario, sólo tienen que mantener el valor de *CR*. Cuando los procesos pasan a estado pasivo, transfieren *CR* al proceso detector. Al recibir un mensaje un proceso activo, si la cantidad de créditos recibida *CR* es igual al valor propio de *CR*, sólo tienen que decrementar sus unidades crédito en uno, `CR:=CR - 1`, así duplican la cantidad de créditos reales. Por el contrario, si el *CR* recibido difiere del *CR* propio, reenvían los créditos recibidos al proceso detector. El proceso detector mantiene registro de los créditos faltantes para finalizar el sistema.

Entonces, el comportamiento de un proceso en el algoritmo es:

```
if (I want to send a message){
   CR := CR + 1;
   Send(message);
}

if (I receive a message){
   if (I am Passive)
         myCR := CRreceived;
   else if (myCR == CRreceived)
         CR := CR - 1;
   else
```

```
        Send (CRreceived) back to the Enviroment;
}
```

Utilizando $D := D - 2^{-CR}$, donde D inicializa en 1 y representa los créditos reales faltantes, se puede decretar la finalización segura del sistema cuando D es igual a cero. Dada las posibilidades técnicas y la versatilidad del algoritmo, variantes y optimizaciones adicionales pueden añadirse para configurar el método a ciertas exigencias y necesidades. Una de ellas es que el proceso detector no necesita hacer dicho cálculo para detectar la terminación.

Por ejemplo, en el sistema existe un conjunto dinámico de valores enteros representativos de modo que podemos utilizarlos para determinar los créditos faltantes. Conoceremos dicho conjunto como `DebtSet`, es decir, el conjunto de débitos que el sistema tiene en circulación, dicho conjunto pertenece al sistema lógico, o bien conocido como `Enviroment`. Cuando el `DebtSet` se encuentre vacío, se entenderá que el sistema no tiene débitos pendientes y todos los procesos han terminado. El `DebtSet` se inicializa con el elemento 0, o sea, `DebtSet = {0}`.

Suponiendo que el proceso coordinador maneja el `DebtSet`, el proceso coordinador también tiene créditos, de hecho, hacemos que empieze con `CR = 0`, luego al mandar el primer mensaje se incrementa a `1`, luego a `2`, y así continuamente.

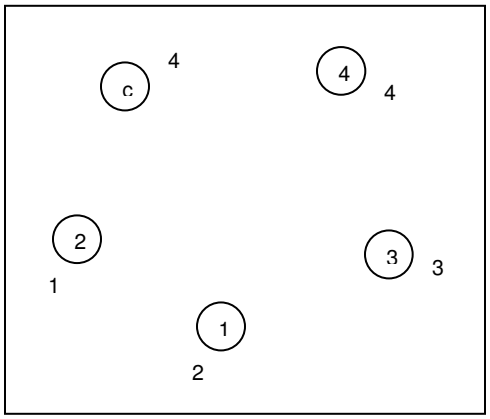
Cuando el `DebtSet` recibe *CRs* de vuelta por finalización de procesos o por redireccionamiento de *CRs* se ejecuta el siguiente algoritmo:

```
K := CR;
while (K is not in DebtSet){
   DebtSet := DebtSet + {K};
   K := K - 1;
}
DebtSet := DebtSet - {K};
if (DebtSet is empty)
   Global Quiescence;    // Algorithm ends.
```

A partir de esto sabemos que:

- Si llega un valor *CR* que ya está en el conjunto, lo saca.
- Si llega un valor *CR* que no está en el conjunto, lo mete.
- La iteración saca del `DebtSet` al primer *K* que encuentre que esté en el conjunto.
- No existen dos copias de un mismo *CR* en el conjunto.

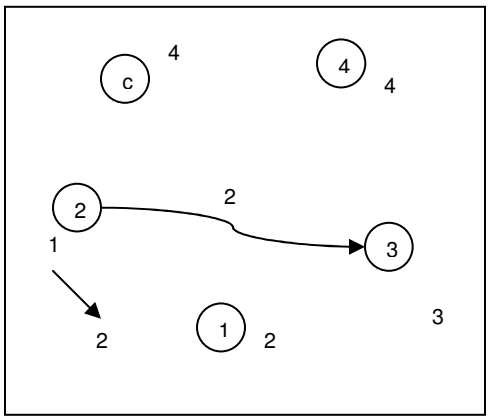
La siguiente secuencia de figuras ilustra la utilización del método, ejemplificando la detección de la terminación a través del DebtSet.



El proceso coordinador *Pc* inicia repartiendo unidades crédito a los procesos que quieren iniciar. Primero inicia con cero créditos y a medida que va repartiendo va incrementando sus créditos.

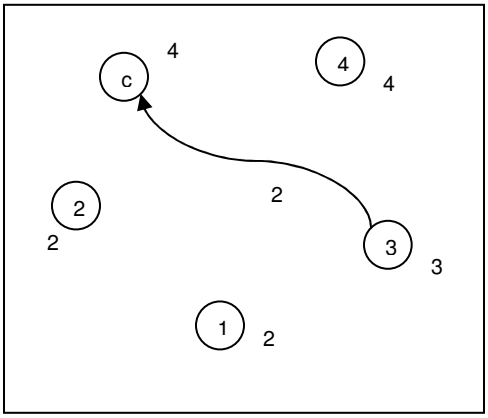
El DebtSet inicia con el elemento {0}. DebtSet = {0}

Note que durante toda la traza, excepto al inicio del algoritmo, el DebtSet mantiene registro de los créditos reales que circulan en el sistema.



El proceso *P2* envía un mensaje al proceso *P3*. Primero incrementa sus unidades crédito y los transmite a *P3*.

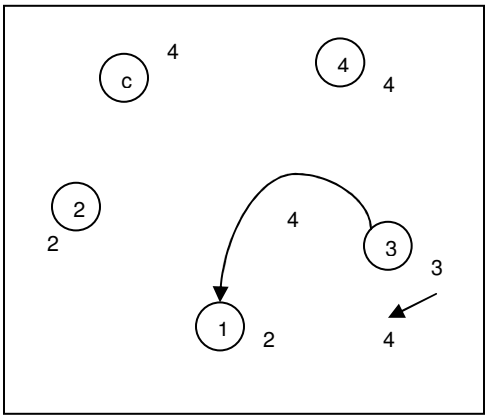
DebtSet = {0}



P3 está activo, y la cantidad de créditos recibidos no es igual a la cantidad de créditos que ya posee, por lo que rechaza los créditos y los reenvía al proceso coordinardor *Pc*. El proceso *Pc* los recibe y ejecuta el algoritmo para introducirlos en el DebtSet.

Según el algoritmo se introducen entonces los elementos {2} y {1} en el DebtSet, y se saca el elemento {0}.

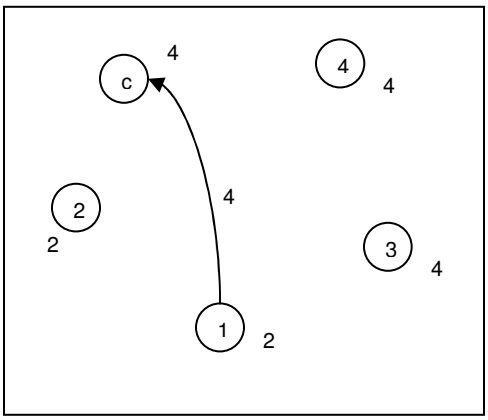
DebtSet = {2, 1}



Note que a partir de ahora el DebtSet registra los créditos circulantes en el sistema. Puede verificarse que la sumatoria del logaritmo binario negativo de 2^{-CR}, donde CR es cada uno de los créditos en circulación, es igual a la sumatoria del logaritmo binario negativo de los elementos del DebtSet.

P3 le envía un mensaje a *P1* por lo que incrementa primero sus créditos y le envía 4 unidades crédito.

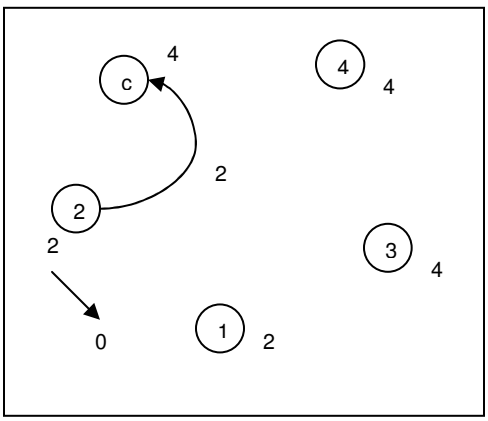
DebtSet = {2, 1}



P1 está activo, y la cantidad recibida es distinta a la cantidad que ya tiene, por lo que rechaza los créditos y los reenvía al coordinador *Pc*.

El coordinador ejecuta el algoritmo correspondiente y se agregan los elementos {4} y {3} al DebtSet, y se saca el elemento {2}.

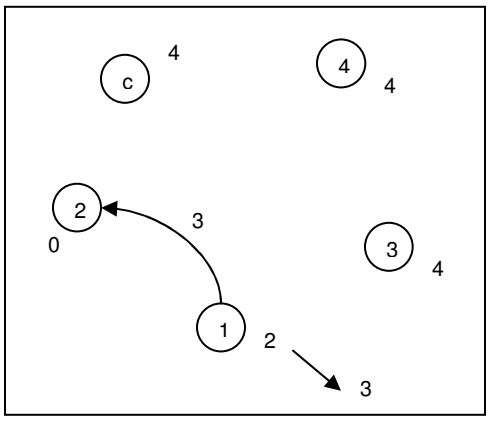
DebtSet = {4, 3, 1}



P2 finaliza entregando sus 2 unidades crédito a *Pc*.

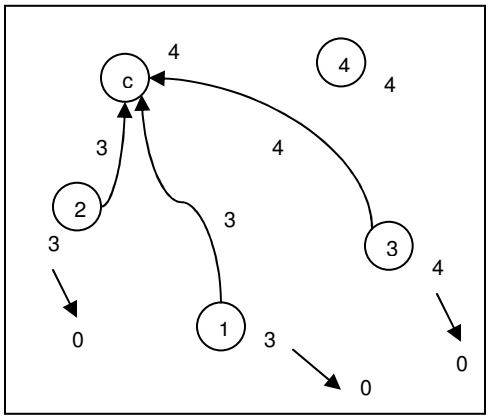
Se introduce el elemento {2} en el DebtSet y se saca el elemento {1}.

DebtSet = {4, 3, 2}



P1 envía un mensaje de aplicación a *P2*. Primero incrementa en una unidad sus créditos y luego envía el mensaje con 3 unidades crédito.

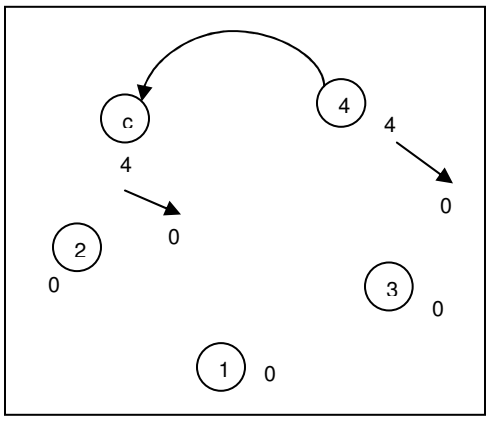
P2 se encuentra Pasivo por lo que recibe los 3 créditos.



Termina *P1* devolviendo sus 3 unidades crédito a *Pc*. Se saca el elemento {3} del DebtSet. DebtSet = {4, 2}

Termina *P2* y devuelve 3 créditos. Se añade el elemento {3} y se saca el elemento {2}. DebtSet = {4, 3}

Termina *P3* y devuelve 4 unidades crédito. Se saca el elemento {4} del DebtSet. DebtSet = {3}



Termina *Pc* y devuelve 4 unidades crédito. Se añade el elemento {4} al DebtSet y se saca el {3}. DebtSet = {4}. *Pc* pasa a estado Pasivo pero mantiene en ejecución el algoritmo de finalización.

Por último termina *P4* devolviendo sus 4 unidades crédito a *Pc*. Se saca el elemento {4} del DebtSet y éste queda vacío.

Se decreta la terminación general del sistema.

1.3- MÉTODOS BASADOS EN ESPERA DE RESPUESTA [DS80] [SF86] [CV90] [DLT92] [DTE07]

Sabemos que el principal problema en la determinación de terminación de un sistema distribuido es la reactivación de nodos que se han decretado en estado pasivo. Cuando se ha iniciado la actividad de detección de terminación, un conjunto de procesos probablemente ha finalizado sus

actividades, pero el conjunto de procesos activos puede mandar mensajes de aplicación reactivando a los procesadores que ya han finalizado ya sea para requerir de sus funciones o para notificar sucesos. El algoritmo de detección basado en Espera de Respuesta está orientado a las actividades en un sistema de computadores. Por lo general, cuando un procesador es reactivado, los procesos locales disparados son tareas consecuentes o sub-tareas de algún proceso foráneo inicial que originó su carga. Al iniciar un sistema de computadores, que estén trabajando de manera paralela o distribuida para llevar a término un gran trabajo o para realizar trabajos de múltiples procesos, todos los procesadores reciben una cierta cantidad inicial de tareas o bien esperan recibir sus porciones de trabajo en algún *pipeline* de procesamiento. En un *pipeline* de procesadores, el producto de salida de un nodo es entrada para otro nodo, en ese caso, la terminación de uno para iniciar otro o la finalización general de todos los nodos es más fácil de determinar. Cuando los procesos son distribuidos e independientes, la detección es distinta.

Si un proceso necesita de otro proceso para realizar una actividad de aplicación, manda un mensaje solicitando sus funciones. Al igual que si debe notificar un cambio o debe iniciar una tarea con la intervención de múltiples procesos, manda un mensaje. El algoritmo de detección con esperas de respuestas se fundamenta en el hecho que buena parte de las veces los procesos retornan valores o notificaciones avisando que las funciones solicitadas han finalizado en buen término. En este método, a todo mensaje enviado le corresponde un acuse de recibo. Un acuse de recibo no necesariamente debe contener algún valor retornado, o algún conjunto de datos que fue generado durante la actividad, puede ser un mensaje vacío. Cuando un nodo procesador envía un mensaje, el procesador receptor no envía la respuesta correspondiente hasta que todos los procesos consecuentes que se originaron finalicen. A partir de este hecho, la detección de terminación puede hacerse.

Por lo general en este tipo de situaciones, cuando el sistema quiere finalizar, un proceso encargado de la terminación inicia un ciclo de consultas para iniciar el cierre del sistema. Es obvio que la terminación no se da de manera inmediata para evitar que los procesos iniciados sin finalizar afecten la consistencia. El proceso detector envía mensajes de control a los procesadores y si éstos han terminado envían una respuesta que indica que están en estado pasivo y no tienen procesos encolados. Los procesadores activos no responden el mensaje de control hasta haber recibido todos las respuestas de los procesos que esperan y hayan finalizado sus funciones.

En vista de esto, sabemos que no es necesario iniciar un ciclo de consulta, es decir, el coordinador no necesita preguntar quiénes han terminado. Tan sólo se requiere que cada proceso notifique que ha terminado y el coordinador llevará el registro de los que están ya pasivos. Note que realmente es imposible que un proceso que sale de la pasividad genere nuevas actividades independientes de la actividad que lo reactivó. Así, no importa que un proceso reactivado haya notificado al coordinador que ya finalizó puesto que su proceso reactivador no decretará su pasividad hasta que reciba todos los acuses de recibo de los procesos que despertó.

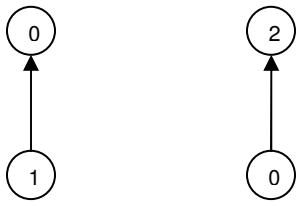
Algunas bibliografías explican este método alegóricamente como un conjunto de árboles de procesos en espera. Cada actividad desencadenada genera un árbol de espera donde la raíz es el proceso que creó la actividad y sus hijos son los procesos foráneos por los cuales la actividad espera. De este modo, en un instante dado, el sistema tiene muchos árboles lógicos correspondientes a las esperas que están ocurriendo.

Cuando un proceso A manda un mensaje de aplicación a otro proceso B, se forma un enlace lógico entre ambos donde B es el hijo de A. Mientras B no responda el mensaje de aplicación con un acuse de recibo, éste sigue siendo hijo de A. Si el mensaje de aplicación generó otras subactividades y B tiene que mandar mensajes a los procesos C y D, C y D van a ser los hijos de B.

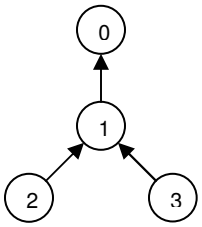
Como es posible que C también requiera de A, es posible que hayan esperas circulares. Como puede verse, es posible la ocurrencia de un abrazo mortal, pero esto ya no es inherente al algoritmo de detección, puesto que el error está en la programación del sistema. Claro está, que si las aplicaciones están bien hechas, A responderá a C eventualmente y la actividad original terminará sastifactoriamente en un tiempo futuro también.

Veamos los árboles de espera generados a partir de la siguiente ocurrencia de hechos:

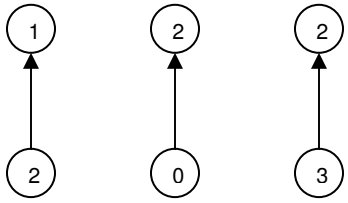
1. N0 dispara un proceso Pr0 en N1
2. N1 dispara un proceso consecuente de Pr0 en N2 y N3
3. N2 dispara un proceso Pr21 en N0
4. N2 dispara un proceso Pr22 en N3



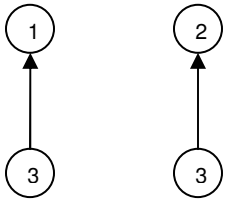
N0 tiene dos árboles de espera, un árbol correspondiente al proceso Pr0, y otro del proceso Pr2. En este último N2 espera por él.



N1 tiene sólo un árbol, donde espera por N2 y N3 y a la vez N0 espera por él. No retornará respuesta hasta que N2 y N3 hayan retornado respuesta.



N2 tiene 3 esperas correspondientes a tres procesos distintos e independientes.



N1 y N2 esperan por N3.

Se debe tomar en cuenta que:

- Cada nodo sólo debería ver la porción del árbol de espera en la que participa, esto logra estructuras más sostenibles en el tiempo.
- Cada árbol debe ser independiente de los demás, para evitar abrazos mortales.
- Deben crearse árboles distintos por cada “padre” a pesar que ambas esperas provienen de un mismo proceso remoto inicial. Así cada espera dura el menor tiempo posible.

CAPÍTULO 2.- DESARROLLO DEL TRABAJO ESPECIAL DE GRADO

2.1.- OBJETIVOS DEL TRABAJO

2.1.1.- OBJETIVO GENERAL

El desarrollo de un sistema de animación que simule la ejecución de los algoritmos distribuidos de detección de terminación para ser utilizado en la enseñanza de éstos.

2.1.2.- OBJETIVOS ESPECÍFICOS

1. Animar algoritmos que siguen un patrón de ejecución multiprocesador en un entorno centralizado/monoprocesador.

2. Simular la ejecución de múltiples procesos que realizan actividades distribuidas y se comunican a través del pase de mensajes.

3. Permitir la configuración del sistema, como la cantidad de procesos que intervienen, el grafo de comunicación, tiempos de ejecución, etcétera. que soporte el proceso de instrucción de los sistemas distribuidos.

4. Programar métodos básicos para la detección de terminación en sistemas distribuidos.

2.2.- EL SISTEMA DESARROLLADO

El sistema programado ejecuta una animación por eventos discretos y orientada a los procesos.

Para efectos de la simulación, a nivel de datos los procesos son sólo tareas que consumen los respectivos ciclos de reloj en cada nodo. Tampoco existe el monitoreo a un socket real, sin embargo la programación de un módulo que realice dicha actividad llevaría el software al nivel de un sistema real, lo que lo diferencia de un sistema corriente de animaciones simples.

Se ha desarrollado el software en el **lenguaje *C*** dada su alta interacción con el nivel de hardware, su manejo de hilos y procesos, su eficiencia y la posibilidad de generar código binario ejecutable.

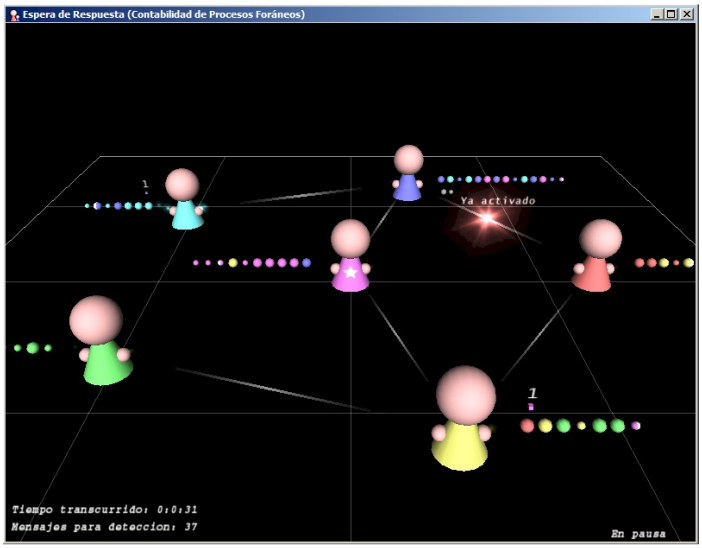


Figura 5.- Animación en ejecución

Casi toda la estructura del software fue codificada completamente desde cero, no se utilizaron librerías de simulación, componentes de desarrollo de terceros ni motores, a excepción de las librerías estándares, la librería gráfica y el generador de números pseudoaleatorios.

Para la propuesta gráfica se utilizó la librería *OpenGL*, con un control de bajo nivel sobre el despligue de los elementos gráficos en pantalla y el cálculo de las transformaciones necesarias para las animaciones.

El generador de números pseudoaleatorios es un *Mersenne Twister*.

La aplicación está desarrollada para plataformas *Windows*, se utilizaron las funciones de la ***Win32 API***, la interfaz de programación de más bajo nivel para desarrollar aplicaciones orientadas a ventanas en la plataforma Windows, maximizando la relación velocidad-tamaño del programa.

ALGUNAS LIBRERÍAS UTILIZADAS

Creación y gestión de hilos:

- Librería: Kernel32.lib
- Cabecera: Winbase.h (incluir Windows.h)

Generación de números aleatorios:

- Librería: randomacof32.lib
- Cabecera: randoma.h

- Disponible en: *www.agner.org*

Gráficos:

- Librería: OpenGL32.lib
- Cabecera: gl.h

- Librería: glu32.lib
- Cabecera: glu.h

- Librería: glut32.lib
- Cabecera: glut.h
- DLL: glut32.dll

Sonidos:

- Librería: Media Control Interface (MCI): Winmm.lib
- Cabecera: mmsystem.h, incluir Windows.h

- Librería: Waveform Audio API: coredll.lib
- Cabecera: Mmsystem.h, incluir Windows.h

2.3.- CARACTERÍSTICAS DEL SISTEMA

Dada la descripción del sistema planteado, podemos determinar cuatro estados esenciales en la aplicación desde el punto de vista del usuario final, así como se diagrama en la **Figura 6**:

1. Un estado inicial en donde se explique el método de detección,
2. un estado de animación y/o simulación,
3. un estado de congelamiento de la animación, y
4. un estado de previa configuración.

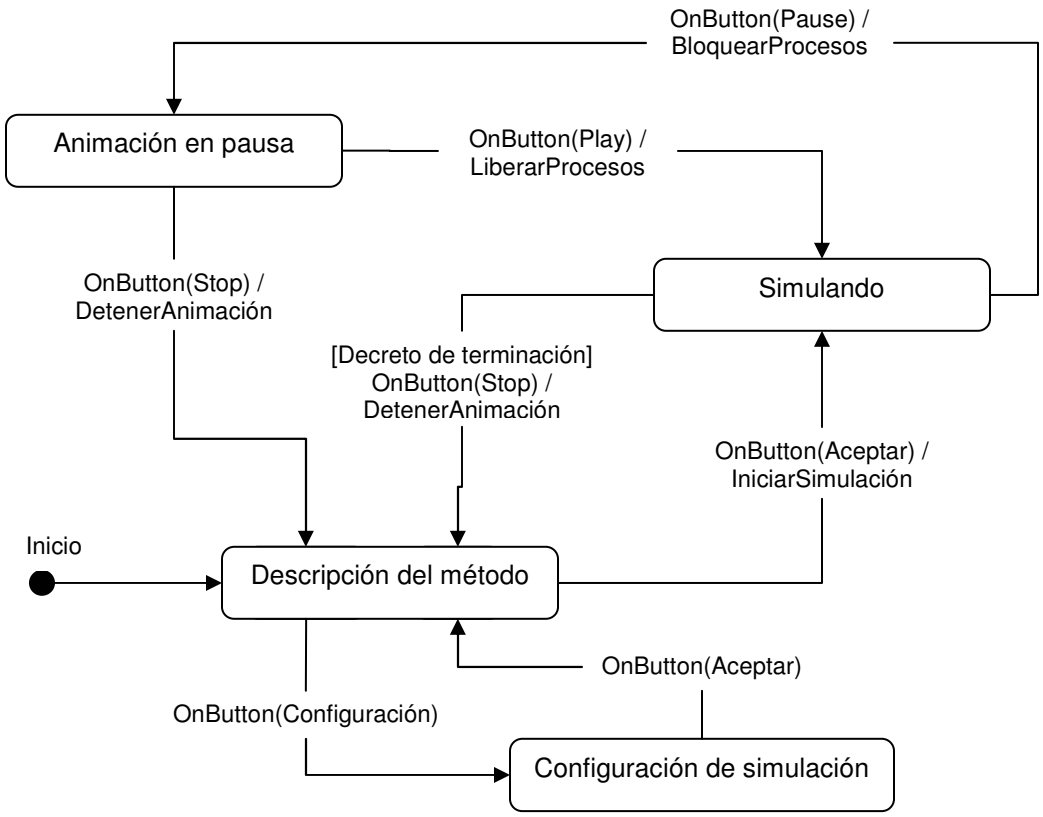


Figura 6.- Estados del sistema

Las estructuras básicas sobre las que se sostiene el sistema simulado son: (1) Una topología de comunicación, (2) un conjunto de nodos de procesamiento y (3) un conjunto de actividades cooperativas y distribuidas a procesar. Para esto, cada nodo de procesamiento trabaja como un módulo de gestión de procesos en el que la culminación de las actividades cooperativas requiere la petición de ayuda a procesadores foráneos a través de la red de comunicación.

Describiendo la simulación desde el punto de vista de los nodos:

- Hay una topología de comunicación entre múltiples nodos procesadores.
- Cada procesador tiene una cola de procesos en memoria y una cola de mensajes entrantes a procesar.
- En cada ciclo iterativo un procesador toma un proceso de su cola y ejecuta su actividad.

- Los procesos de un nodo pueden ser propios o pueden ser procesos disparados por otros nodos.
- Los procesos que necesitan ayuda foránea disparan procesos en otros nodos procesadores.
- Los procesos pueden exportar trabajo a otros nodos, disparando también procesos foráneos.
- Los procesos importados pueden acarrear la devolución de un mensaje.
- Finalizar un proceso puede llevarse varios ciclos de iteración.
- Ningún nodo conoce la topología completa de la red, sólo conoce a sus vecinos.

La **Figura 7** describe un ciclo iterativo de un nodo procesador.

En cuanto al procesamiento de mensajes:

- En cada ciclo un nodo procesa un mensaje de la cola de entrada.
- Hay dos clases de mensajes: Mensajes ordinarios y mensajes de control. Los mensajes de control son mensajes de activación iniciales o mensajes concernientes al algoritmo de terminación.
- Los mensajes ordinarios que acarrean actividades disparan procesos en los nodos receptores. Los otros mensajes ordinarios son respuestas a actividades esperadas.
- Cuando un nodo toma un proceso que espera un dato lo suelta y no sigue su ejecución hasta recibir el mensaje con el dato esperado.

La **Figura 8** muestra el conjunto de actividades necesarias al procesar un mensaje entrante.

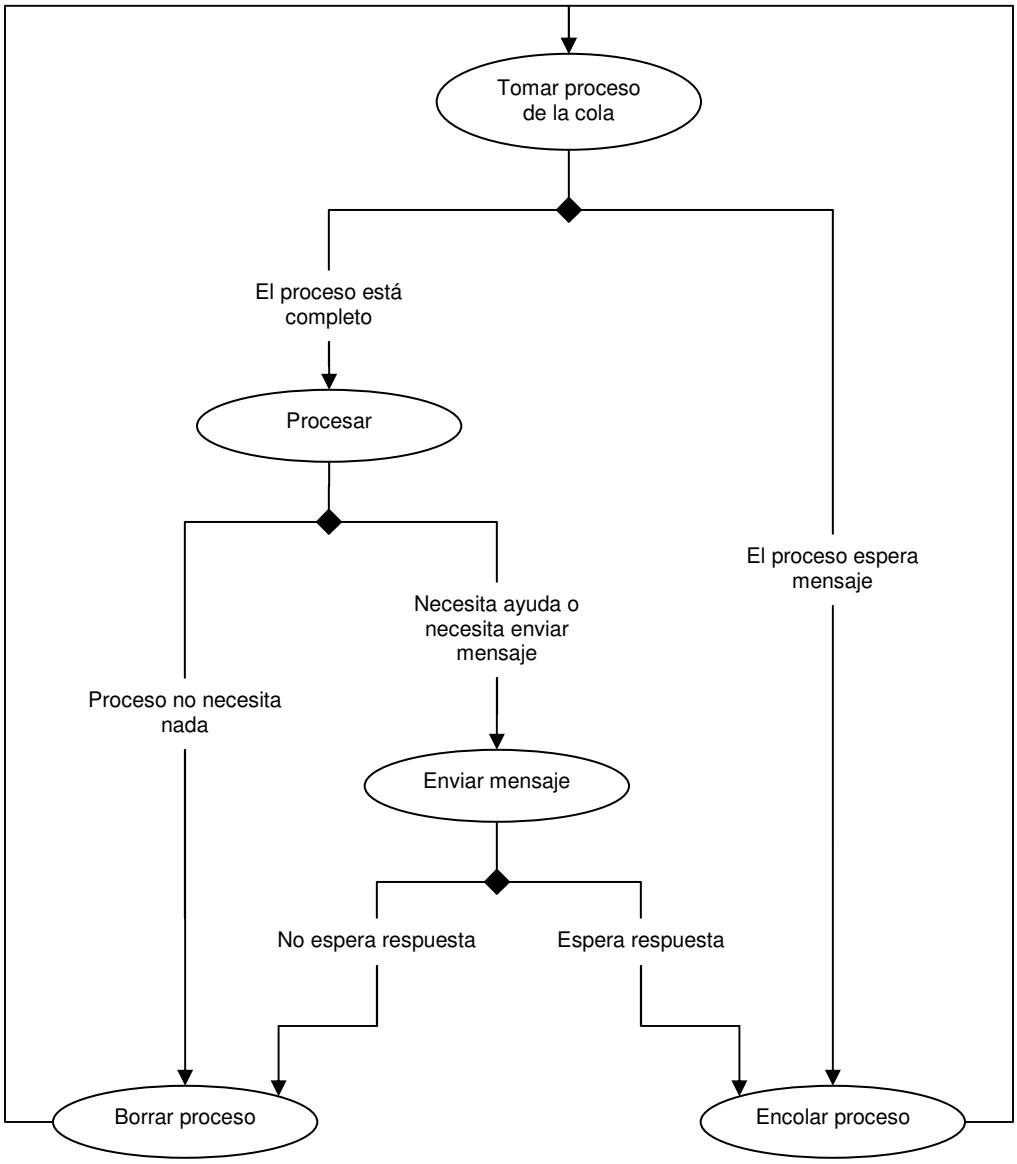


Figura 7- Ciclo de procesamiento de un nodo.

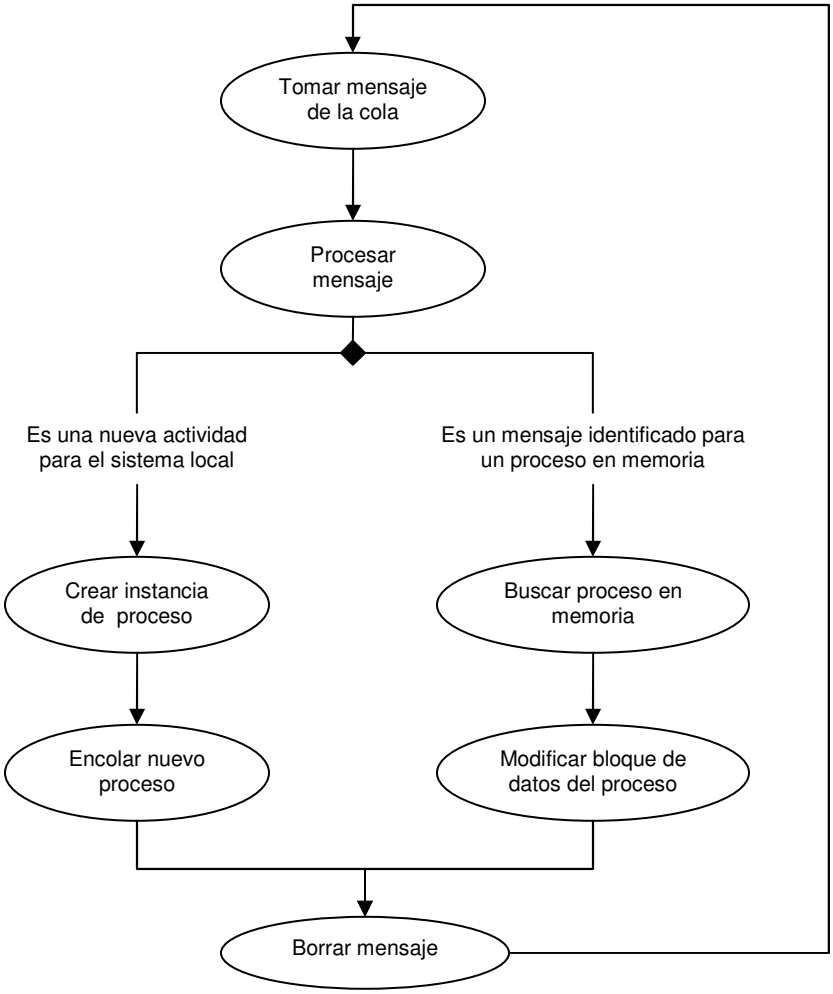


Figura 8- Diagrama de actividades para procesar la cola de mensajes en un nodo.

2.4.- ESTRUCTURAS DE DATOS

Las tres estructuras básicas de datos definen a:

1. Los nodos procesadores que conforman toda la red del sistema simulado.
2. Los procesos cargados en memoria de cada nodo procesador.
3. Los mensajes tanto en tránsito como en las colas de mensajes.

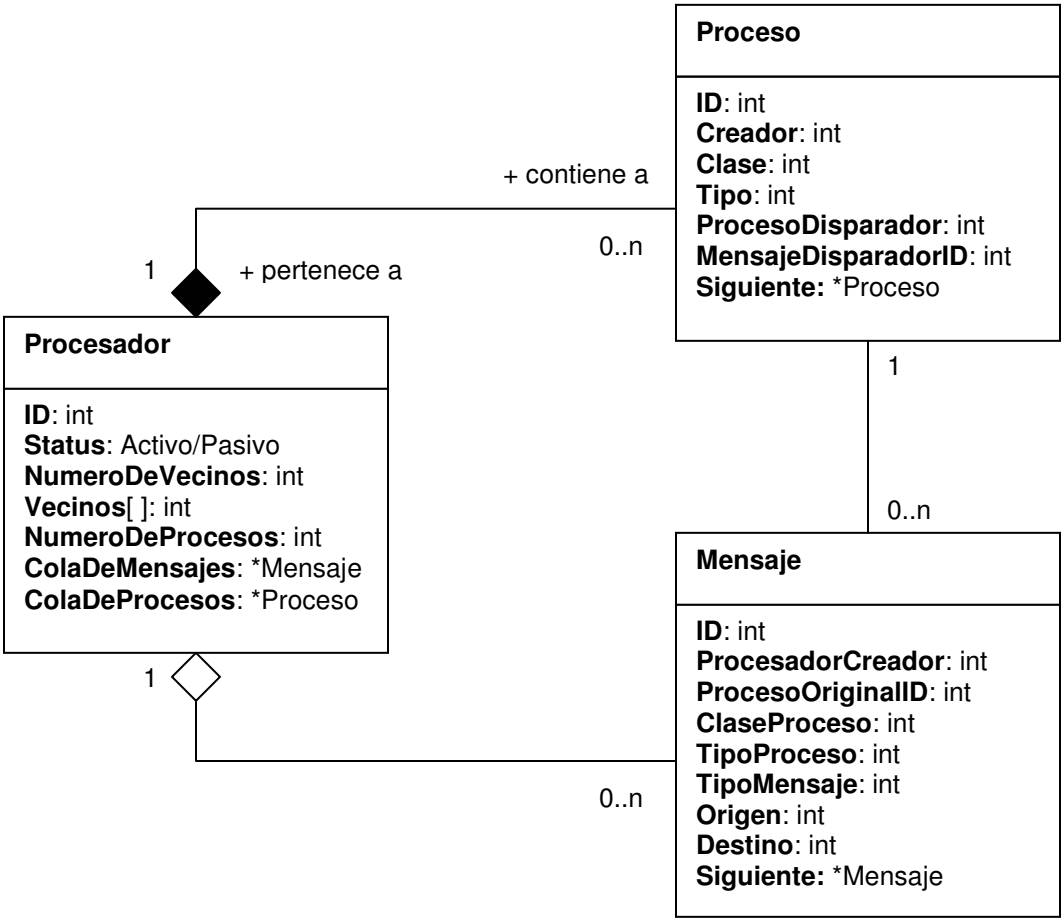


Figura 9.- Forma básica de las estructuras esenciales y sus relaciones.

2.5.- IMITAR PROCESADORES/PROCESOS CON HILOS

Lo primero que ha de tomarse en cuenta al programar un sistema de múltiples ejecutores de tareas, son los procesos, siendo lo primero en decidirse la unidad de ejecución que se va a emplear: *procesos* o *hilos*. Tanto unos como otros son unidades de trabajo utilizados para paralelizar un sistema y aunque pudiera confundirse ambos términos como casi lo mismo dependiendo también del contexto literario en el que se utilizan (con la llegada de tecnologías como *Hyper-Threating*), las diferencias técnicas son determinantes. Los procesos no comparten su espacio de direcciones a diferencia de los hilos de un mismo proceso y esa es una distinción más que suficiente. Por tanto, la comunicación entre procesos siempre es más costosa porque requiere cambios de contexto.

En términos de soluciones de software de más alto nivel, el proceso es más una unidad lógica al momento de diseño, una categorización o demarcación de un conjunto de funciones para la construcción de la arquitectura de solución. En tanto que un hilo sería una de las unidades específicas de funcionamiento de muy bajo nivel que llevan a cabo las tareas de un proceso.

Debe quedar claro que en una animación monoprocesadora como la nuestra lo que requerimos son ***hilos*** en un mismo contexto, aunque simulemos que son ***entes*** que se comunican a través del pase de mensajes, representando el transporte de mensajes también con un ***hilo***.

Al crear un simple programa, creamos un proceso con un hilo de ejecución. Para realizar una animación como la nuestra existen dos posibilidades:

1. Utilizar un solo hilo que realice todas las actividades del sistema, representando eventualmente todos los entes/procesos del sistema distribuido y el pase de mensajes, que represente con cada iteración el envejecimiento del sistema.

2. Programar la generación de múltiples hilos de ejecución que representen los entes/procesos del sistema y envejecer el sistema en intervalos de tiempo controlado.

En animación y simulación, ésta última opción es preferida en cuanto a modularidad y control de tiempo se refiere, a pesar que requiere mayor cuidado al programar y consume más recursos y tiempo de procesador al ejecutarse.

2.6.- EL AMBIENTE DE EJECUCIÓN

En cuanto al sistema representado, otro elemento esencial es la ***cola de mensajes***, una estructura que aparente las colas de los medios de comunicación, donde los mensajes se alisten y puedan ser transmitidos, no debe utilizarse la cola de mensajes del framework de aplicación que administraría el sistema operativo, ya que ésta es inmediata y no debe ser administrada por el sistema simulado. Para transmitir los mensajes debe haber el hilo de ejecución que actúe como ***transportador de mensajes*** recogiendo cada mensaje de la cola y transfiriéndolo a lo largo del sistema hasta el buzón de entrada del punto destino. La transmisión de mensajes debe hacerse de manera gradual, permitiendo la ocurrencia de otros sucesos durante las transmisiones. Lo ideal es que el mensajero de mensajes sea el mismo hilo que dibuja la escena, así no hay discrepancias entre lo que sucede y lo que visualmente se muestra, este hilo de ejecución debe ser de máxima prioridad para que los cambios de estado y ocurrencias de las escena se reflejen de inmediato y no haya saltos entre escenas.

Para una adecuada animación el hilo principal de ejecución o hilo de ambiente, debe componerse de dos partes, la de renderizado y la de mensajería. En el renderizado todos los elementos de escena se despliegan, a partir de la información de cada proceso se dibuja algo en pantalla, cada proceso debe guardar en su segmento de datos además de sus variables de ejecución, banderas de estado y colas de mensajes, información de su posición en pantalla y los mensajes de animación de usuario. El problema en la simulación aquí es que cuando el número de procesos a animar es muy grande, la espera por las secciones críticas se hacen muy largas, además que cada proceso tiene que esperar por el o los únicos procesadores del sistema centralizado. En el segmento de mensajería, el hilo representa las vías de comunicación. El movimiento de un mensaje en escena se calcula con la interpolación lineal del proceso origen y el proceso destino: $P_{origen} * t + P_{destino} * (t - 1)$, donde *t* discretiza el desplazamiento. Otra tarea del ambiente de ejecución es velar por la exclusión mutua de las secciones críticas, en este caso, las colas de mensajes y los buzones de entradas son las estructuras más concurridas y volátiles.

Dado que la ejecución debe considerar el aspecto de animación, los tiempos de ejecución real no son iguales a los simulados, es decir, un proceso puede terminar muchas más tareas de lo que una animación legible podría mostrar, por tanto en cada iteración en el tiempo de vida de un proceso, se le obliga al proceso a descansar y liberar el procesador en su turno. El punto positivo de esto es que

el procesador queda libre la mayor parte del tiempo y el hilo graficador de la escena no espera tanto tiempo por utilizar su turno.

Podemos separar entonces, o definir teóricamente, a los hilos de ejecución en módulos de ejecución, que no necesariamente tienen que estar delimitados a nivel de codificación e implementación, pero que pueden servir de guía para la comprensión del código base durante el entendimiento. Cinco grupos de características funcionales que crean el ambiente de ejecución y los procesos circunstantes que se muestran en la **Figura 10**. Una primera porción que corresponde al hilo principal de la aplicación y se encarga de inicializar el ambiente imitado, iniciar los parámetros visuales básicos y también cerrar el ambiente de animación y borrar las variables correspondientes cuando la representación termina.

Una segunda porción clasificable es la sección de visualización, encargada de las labores de pintar las interfaces de usuario y por tanto de las interacciones inmediatas entre el sistema y el usuario, manejar los parámetros de la animación: proyecciones de perspectiva, límites de visualización, variables de escena, escritura en memoria de gráficos y otros elementos multimedia.

También están los hilos de simulación, o procesadores simulados, paralelamente al hilo conductor de mensajes. Y por último la sección que sirve de interfaz para administrar los recursos críticos en el falso sistema, gestionando la adición, lectura y eliminación de mensajes de los buzones de entrada y de la lista de mensajes en transmisión.

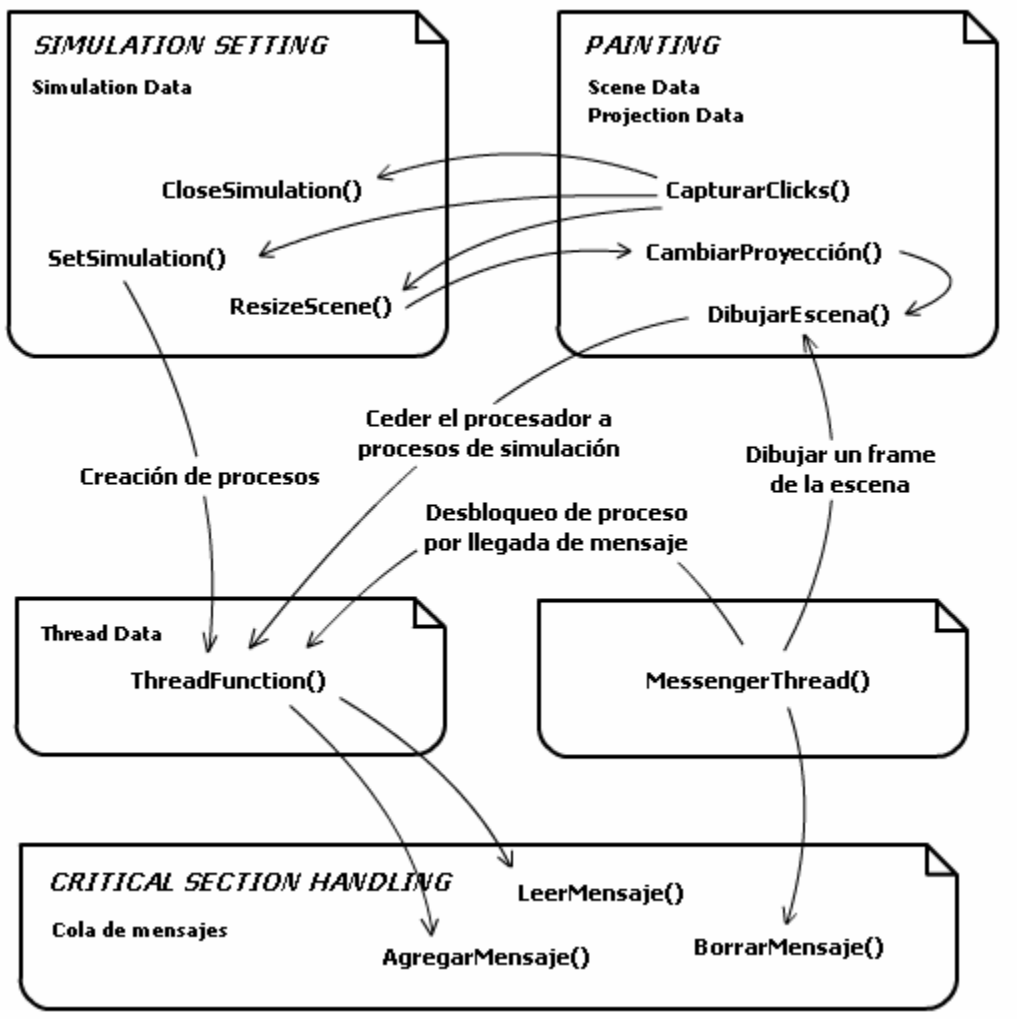


Figura 10.- Bloques lógicos de ejecución del sistema

2.7.- IMPLEMENTACIÓN

2.7.1.- DEFINICIÓN FORMAL DE LAS ESTRUCTURAS

Las estructuras para describir los elementos **Procesador**, **Proceso** y **Mensaje** son:

```
struct PROCESSOR{
	int ID;				// ID de procesador.
	bool status;			// Activo o Pasivo.
	GLfloat r, g, b;		// Color.
	GLfloat x, y, z;		// Posición.
```

```
        int ProcessCounter;        // Para darle IDs a mis procesos propios.
        int MessageCounter;        // Para darle IDs a mis mensajes.
        int numProcesses;          // Cantidad de procesos en el pool.
        int numNeighborgs;         // Cantidad de vecinos que tengo.
        int neighborg[];           // Mis vecinos.
        int numNeighborgsVisited;  // Cantidad de vecinos que han recibido un
                                   // GLOBAL QUIESCENCE MESSAGE (GQM).
        bool neighborgVisited[];   // Indica si el vecino fue visitado por un
                                   // mensaje de terminación (GQM).
        bool neighborgActivated[];// Indica si el vecino recibió un mensaje de activación.
        bool bActiveVecinos;       // Para no mandar infinitamente mensajes de activación.
        bool bYaHeMandadoMsjTerminación; // Para rechazar mensajes de terminación.
        int MyActivator;           // El vecino que me envió la activación.
        int EsperaPorMi;           // Proceso que me envió un mensaje de terminación y yo
                                   // le responderé una confirmación.
        CRandomMersenneA *RanGen;  // Generador de número aleatorios.
        ACTIVITY *ProcessPool;     // Apunta al proceso que se está tratando.
        MESSAGE *Inbox;            // Primer mensaje de la cola.
        MESSAGE *lastMessage;      // Último mensaje de la cola.
};


struct PROCESS{
        int ID;                    // ID de proceso.
        int creator;               // Nodo creador (El mismo valor para todo proceso
                                   // foráneo que dispara).
        int clase;                 // Clase de Proceso: Ordinario o De control.
        int tipo;                  // Tipo de Proceso.
        int sender;                // Nodo que disparó el proceso.
        int MessageID;             // Mensaje que disparó el proceso.
        struct DATA *Data          // Bloque de datos.
        struct PROCESS *next;      // Siguiente proceso en la cola.
        struct PROCESS *previous;  // Proceso previo en la cola.
};


struct MESSAGE{
        int ID;                    // ID de mensaje.
        int ProcessCreator;        // Nodo que creó el proceso que generó el mensaje.
        int OriginalProcessID;     // Proceso primario que origina el mensaje.
        int claseProceso;          // Clase de Proceso: Ordinario o de control.
        int tipoProceso;           // Tipo de Proceso.
        int sender;                // Nodo remitente.
        int tipoMensaje;           // Tipo de mensaje: Solicitud de ayuda, Respuesta a
                                   // peticion de ayuda, etc.
        int origen;                // ID de nodo origen.
        int destino;               // ID de nodo destino. Sólo se utiliza mientras está en
                                   // tránsito.
        struct DATA *Data          // Bloque de datos.
        float t;                   // Para la interpolación mientras está en tránsito.
        GLfloat r, g, b;           // Color.
        struct MESSAGE *INext;     // El siguiente mensaje si está en bandeja de entrada.
        struct MESSAGE *CLNext;    // El siguiente mensaje si está en tránsito.
   };
```

2.7.2.- ALGORITMO BÁSICO DE CADA NODO PROCESADOR

El código base de cada nodo procesador de los métodos implementados consta de un bucle infinito. Una vez iniciado un nodo, el hilo de ejecución entra en un bucle lógico donde constantemente verifica la llegada de mensajes y procesa los procesos cargados en la cola. En cada ciclo se toma un solo proceso de la cola y se trabaja su actividad, finalizado su tiempo de ejecución, el ciclo procede a tomar un mensaje de la cola de entrada y procesa el mensaje. Los ciclos se bloquean una vez que no hayan procesos en memoria ni mensajes en la bandeja de entrada, ahí esperan a recibir más mensajes ordinarios de aplicación o mensajes de terminación.

```
while (true)
{
	Prc = ColaDeProcesos; // Tomar un proceso de la cola.

	if (Prc){	// Si no es NULL.

		switch(Prc.Clase){  // ¿Proceso Ordinario o De Control?
			case PROCESO_ORDINARIO:{
				// Procesar
				// ...

				// Apuntar al siguiente proceso.
				ColaDeProcesos = Prc.Siguiente;

				if (Prc.IsComplete())	// ¿Finalizado?
					// Borrar el proceso.
					Delete(Prc);
				break;
			}

			case PROCESO_DE_CONTROL:{
				switch (Prc.Tipo){  // ¿Qué tipo de Proceso de Control es?
					case ACTIVATION_PROCESS:{  // Proceso para repartir
									// activaciones.
						if (!Treat_an_ACTIVATION_PROCESS(Prc))
							Delete(Prc);
						else
							// Tomar el siguiente proceso.
							ColaDeProcesos = Prc.Siguiente;
						break;
					}	// cierra case ACTIVATION_PROCESS

					default:{	// Otro tipo de Proceso de Control.
						// Procesar
						// ...
						// Delete(Prc);
						// ColaDeProcesos = Prc.Siguiente;
						break;
					}
				}	// cierra switch (Prc.Tipo)

				break;
			}	//	cierra case PROCESO_DE_CONTROL

		}	// cierra switch (Prc.Clase)
	}

	//=============================================================================
```

```
// Ahora procesar algún mensaje de la bandeja de entrada.


if (ColaDeMensajes)       // Tengo mensajes entrantes.
      Msg = ColaDeMensajes;     // Tomo el primero.

if (Msg){

      switch (Msg.tipo){        // ¿Qué tipo de mensaje es?

            case RESPONSE_MESSAGE:{          // Es un mensaje esperado.

                  UpdateProcess(Msg); // Actualizar bloque de datos del proceso.
                  break;
            }
            case ACTIVITY_MESSAGE:{    // No es un mensaje esperado, creo un nuevo
                                       // proceso.

                  CreateProcessFromMessage(Msg);
                  break;
            }
            case ACTIVATION_MESSAGE:{ // Es un mensaje de activación.

                  if (ActiveVecinos)  // Ya lo he recibido antes.
                        // Rechazar el mensaje.
                        SendAMessage(Msg.sender,REJECT_ACTIVATION_MESSAGE);
                  else{
                        CreateProcessFromMessage(Msg);
                        // Anotar a quién debo responder.
                        myActivator = Msg.Sender;
                  }

                  // Marcar vecino como Activado.
                  neighborgActivated[Msg.Sender] = true;
                  break;
            }
            case RESPONSE_ACTIVATION_MESSAGE:{

                  UpdateProcess(Msg); // Actualizar bloque de datos del proceso.
                  break;
            }
            case REJECT_ACTIVATION_MESSAGE:{

                  UpdateProcess(Msg); // Actualizar bloque de datos del proceso.
                  // Marcar el vecino como Activado.
                  neighborgActivated[Msg.Sender] = true;
                  break;
            }
            case GLOBAL_QUIESCENCE_MESSAGE:{

                  if (!Process_a_GLOBAL_QUIESCENCE_MESSAGE(Msg))
                        return 0;     // El nodo retorna/finaliza.
                  break;

            }
            case NOTIFIED_MESSAGE:{

                  if (!Process_a_NOTIFIED_MESSAGE(Msg))
                        return 0;     // El nodo retorna/finaliza.
                  break;
            }
      }     // cierra switch
```

```
                ColaDeMensajes = Msg.Siguiente;         // El siguiente ahora será el primero.
                Delete(Msg);                                    // Borrar el mensaje procesado.

        }       //cierra if (Msg)

        //=============================================================================


        // SI NO QUEDAN PROCESOS NI MENSAJES.
        if ((ColaDeProcesos==NULL)&&(ColaDeMensajes==NULL)){

                Status = PASIVE;
                // Bloquearme hasta recibir algún mensaje.
                WaitFor(MessageEvent());
        }

}       // cierra while (true)



/*****************************************************************************************/


int Processor::Process_a_GLOBAL_QUIESCENCE_MESSAGE(Msg MESSAGE){

        // Marcar el vecino para no enviarle GLOBAL_QUIESCENCE_MESSAGE.
        neighborgVisited[Msg.Sender] = true;

        // Incrementar número de vecinos visitados.
        numNeighborgsVisited += 1;

        if (Notified){
                // Devolver un NOTIFIED_MESSAGE.
                SendAMessage(Msg.sender,NOTIFIED_MESSAGE);
        }
        else{  // Primera vez que recibe un GLOBAL_QUIESCENCE_MESSAGE
                Notified = true;
                // Anotar a quién debo devolver un NOTIFIED_MESSAGE.
                myNotifier = Msg.sender;

                if (numNeighborgsVisited==numVecinos){
                        // Sólo entran aquí los que tienen un solo vecino.

                        // Notificar que todos mis vecinos han sido visitados.
                        SendAMessage(myNotifier,NOTIFIED_MESSAGE);

                        // Borrar el mensaje antes de finalizar.
                        Delete(Msg);

                        // El nodo retorna/finaliza.
                        return 0;
                }

                else { // Desconozco si todos mis vecinos han sido visitados
                                // por un GLOBAL_QUIESCENCE_MESSAGE

                        // Propagar el GLOBAL_QUIESCENCE_MESSAGE
                        int j = 0
                        while (neighborgVisited[j])
                                j += 1;
                        SendAMessage(j,GLOBAL_QUIESCENCE_MESSAGE);
                }
        }
```

```
	return TRUE;
}

/*****************************************************************************/


int Processor::Process_a_NOTIFIED_MESSAGE(Msg MESSAGE){

	// Marcar el vecino para no enviarle GLOBAL_QUIESCENCE_MESSAGE.
	neighborgVisited[Msg.Sender] = true;

	// Incrementar el número de vecinos visitados.
	numNeighborgsVisited += 1;

	if (numNeighborgsVisited==numVecinos){

		if (ID!=COORDINATOR_ID){   // No soy el coordinador.
			// Notificar que todos mis vecinos han sido visitados.
			SendAMessage(myNotifier,NOTIFIED_MESSAGE);
		}
		else { // Es el coordinador
		}

		// Borrar el mensaje.
		Delete(Msg);

		// El nodo retorna/finaliza.
		return 0;
	}

	else{  // Posiblemente aún hay vecinos que no han recibido un
		// GLOBAL_QUIESCENCE_MESSAGE

		// Propagar el GLOBAL_QUIESCENCE_MESSAGE
		int j = 0;
		while (neighborgVisited[j])
			j += 1;
		SendAMessage(j,GLOBAL_QUIESCENCE_MESSAGE);
	}

	return TRUE;
}

/*****************************************************************************/

int Processor::Treat_an_ACTIVATION_PROCESS(Prc PROCESS){

	if (!ActiveVecinos){
		// Propagar ACTIVATION_MESSAGE
		SpreadActivationMessage();
		ActiveVecinos = true;
	}
	else{  // Ya active a mis vecinos.
		if (numProcess==1){ // Es el único proceso que tengo en la cola.

			// Hacer reportes aquí si es necesario
			// ....

			if (Prc.IsComplete()){ // Ya recibí todos los
					// RESPONSE_ACTIVATION_MESSAGE que esperaba.

				if (ID!=COORDINATOR_ID){   // No soy el coordinador
					// Notificar que ya terminé.
```

```
                                SendAMessage(myActivator,RESPONSE_ACTIVATION_MESSAGE);
                        }
                        else{  // Soy el coordinador
                                // Iniciar terminacion.
                                SendAMessage(neighborg[myFirstNeighborg],
                                                GLOBAL_QUIESCENCE_MESSAGE);
                                Notified = true;    // Rechazar todos los
                                                    // GLOBAL_QUIESCENCE_MESSAGE.
                        }
                        // Borrar el proceso.
                        return 0;
                }
                else{  // Faltan RESPONSE_ACTIVATION_MESSAGEs.
                }
        }
    }
    return TRUE;
}
```

2.7.3.- PROGRAMACIÓN DE LOS MÉTODOS

2.7.3.1- MÉTODO DE PROPAGACIÓN DE CONSULTA

Sabemos que el método consiste en propagar una consulta de estado en todo el sistema. Un nodo previamente designado inicia un ciclo de consultas y declara terminación del sistema si el ciclo finaliza. El intercambio de mensajes entre procesos remotos provoca la nulidad del ciclo de consultas vigente, por lo que el ciclo concluye con un mensaje alterado de vuelta al nodo iniciador de la detección.

Podemos minimizar el tiempo que tardamos en detectar la terminación si logramos evitar que cada ciclo fallido concluya, pues esperar a que el nodo coordinador reinicie la consulta es necesario que un mensaje alterado regrese por alguna vía, en el peor de los casos sería después de recorrer todo el sistema. Lo ideal sería que cada ciclo fallido se elimine una vez se sepa que no notificará una detección positiva.

Como no es posible conocer todos los mensajes transmitidos en el sistema por todos los nodos, cualquier nodo que envíe mensajes podría reiniciar el ciclo de consultas. De esta forma la propagación vuelve a empezar desde el instante en que una propagación no concluirá con terminación.

Las **Figuras 11** y **12** muestran los diagramas de actividades concernientes a la implementación de este método.

Cada nodo recibe el mensaje especial y dispara un proceso de control. El proceso de control se mantiene en cola mientras la propagación actual no haya finalizado o el nodo se haya enterado que el ciclo fue reiniciado.

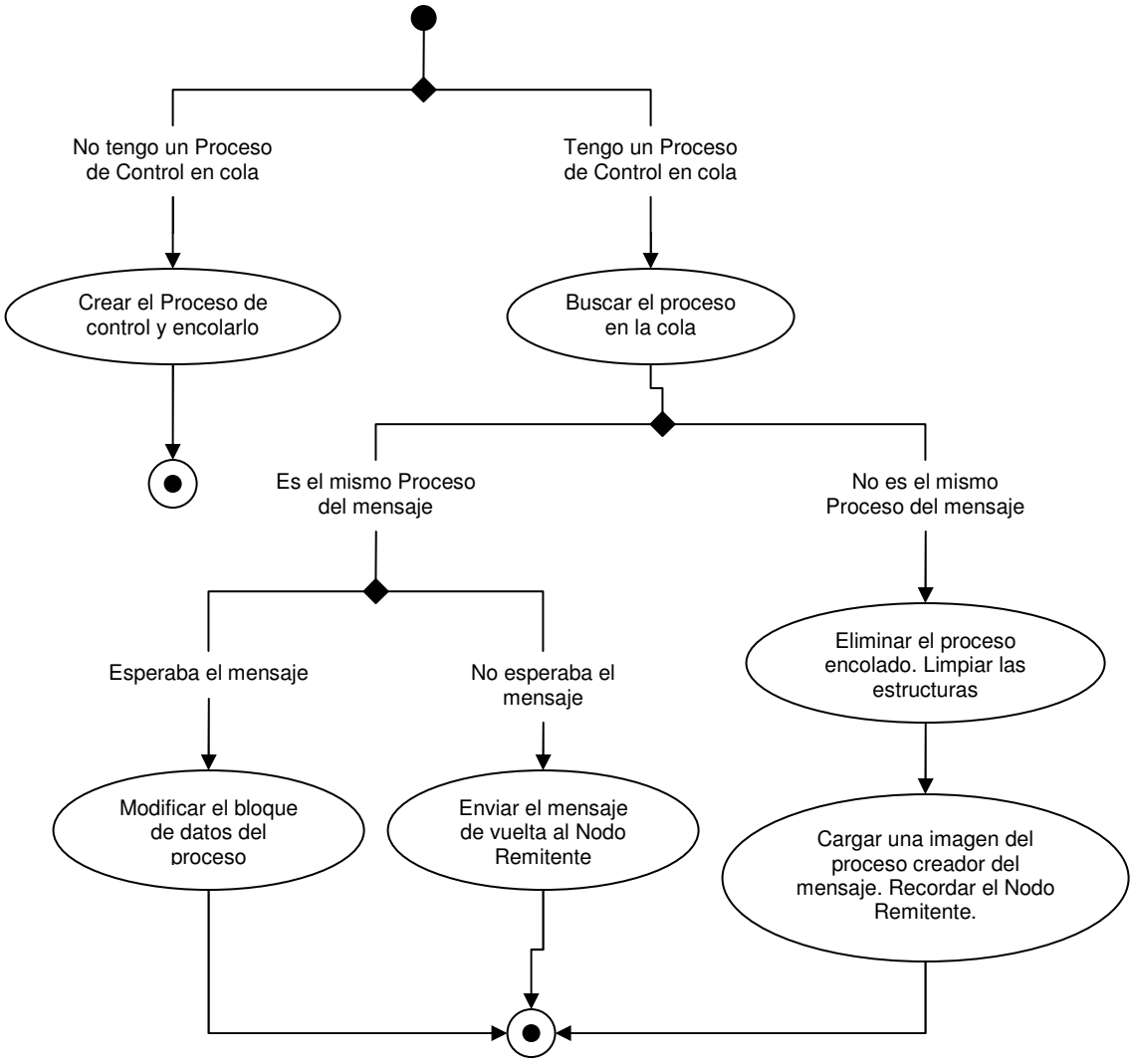


Fig. 11.- Al recibir un mensaje de propagación de consulta.

Una vez finalizados todos los procesos ordinarios locales, cualquier nodo poseedor de la actividad de control puede reiniciar el ciclo de consultas si ha mandado mensajes de aplicación, sino simplemente sigue propagando la consulta o se bloquea hasta recibir mensajes de control.

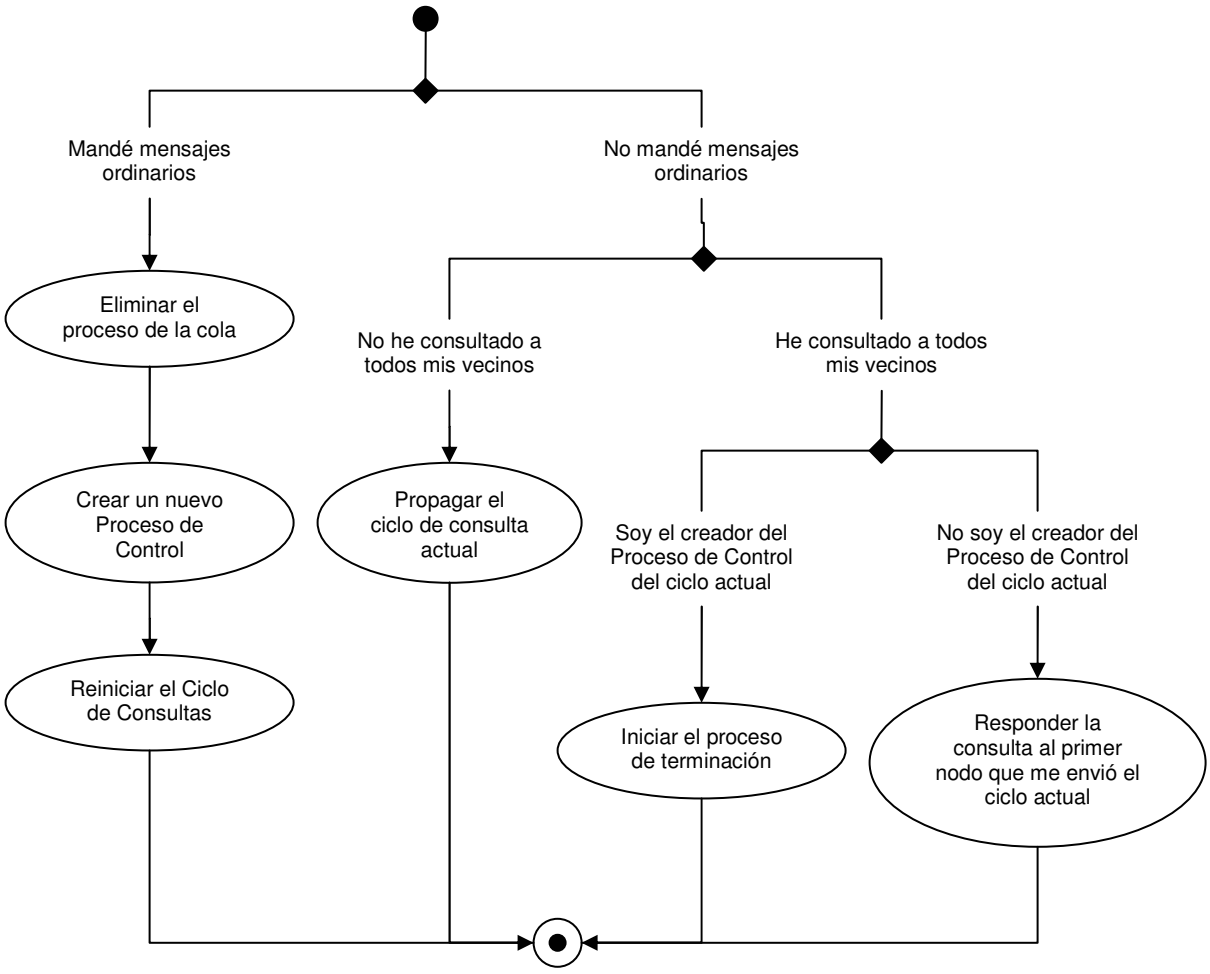


Figura 12.- Actividades al procesar el proceso de control en memoria.

2.7.3.2- MÉTODOS BASADOS EN CRÉDITO

La distribución y recuperación puede realizarse de dos maneras, una utilizando créditos reales y la otra a través del logaritmo binario negativo de la cantidad de créditos reales.

La primera forma requiere una lógica simple dado que sólo se fracciona un valor real al querer mandar un mensaje y se adiciona cuando se recibe. Los problemas con esta forma de trabajar son los redondeos.

Para evitar los redondeos con acarreo de errores, utilizamos el logaritmo binario negativo para trabajar con unidades enteras en vez de valores reales. Resumiendo las matemáticas inherentes, el método se reduce a instrucciones muy simples:

Cuando se quiere mandar un mensaje,

- se le adiciona 1 al crédito y se envía el mensaje con copia del nuevo crédito.

Cuando se recibe un mensaje,

- si el crédito recibido coincide con el crédito local, se le resta 1 al valor y ése es el nuevo crédito;
- pero si el crédito recibido es distinto al local, no puede operarse y se manda al nodo coordinador.

Como esto último no es una operación eficiente dado la cantidad de mensajes que puede recibir un nodo, resolvemos el problema creando en cada nodo procesador una lista de ítems de crédito, así, cuando se recibe un valor de crédito que no puede operarse con ningún crédito local, simplemente se acumula en una cola. Al haber entonces una cola de créditos, cuando un proceso local manda un mensaje se toma un ítem de crédito de la cola en vez de picar el crédito total local en porciones equitativas.

Al enviar un mensaje un nodo ejecuta la secuencia de actividades que se diagrama en la **Figura 13**. La **Figura 14** contiene las actividades a realizar al recibir un mensaje foráneo.

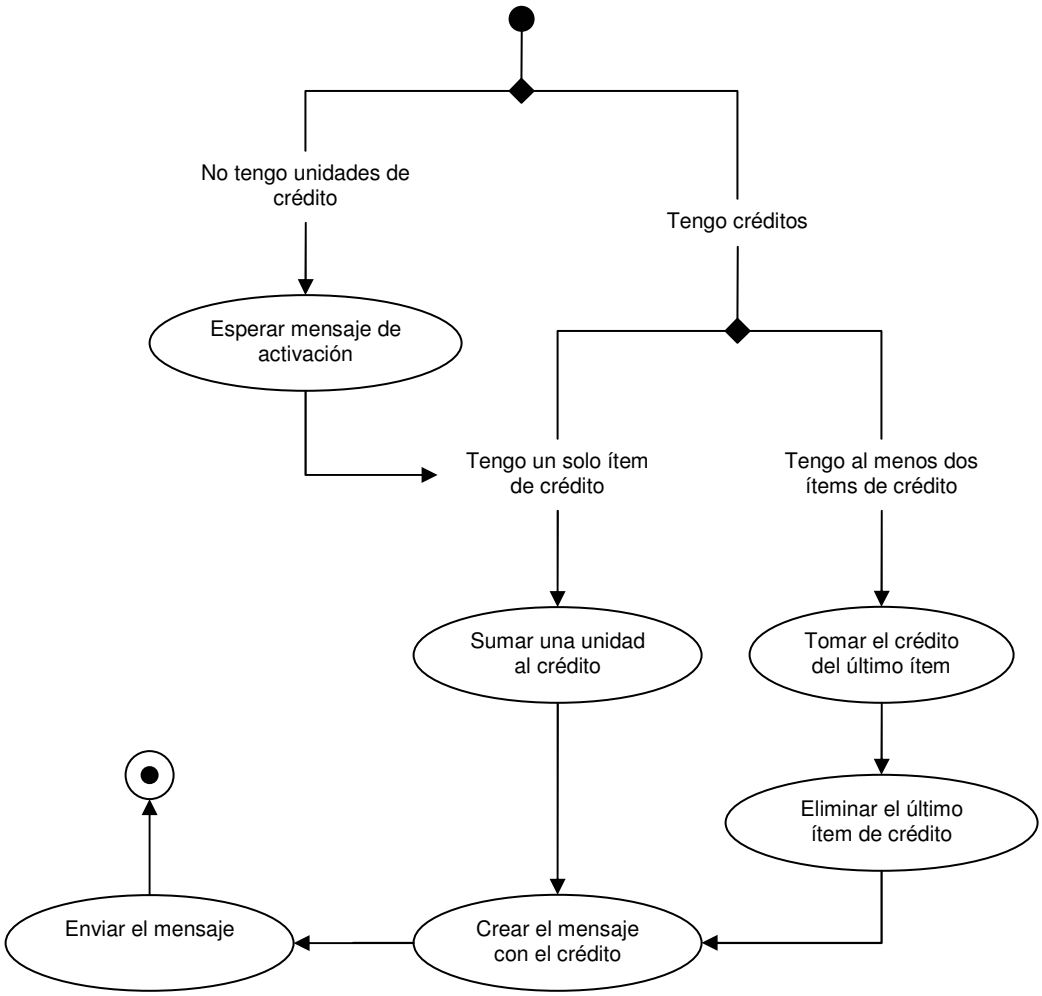


Figura 13.- Al enviar mensajes con créditos.

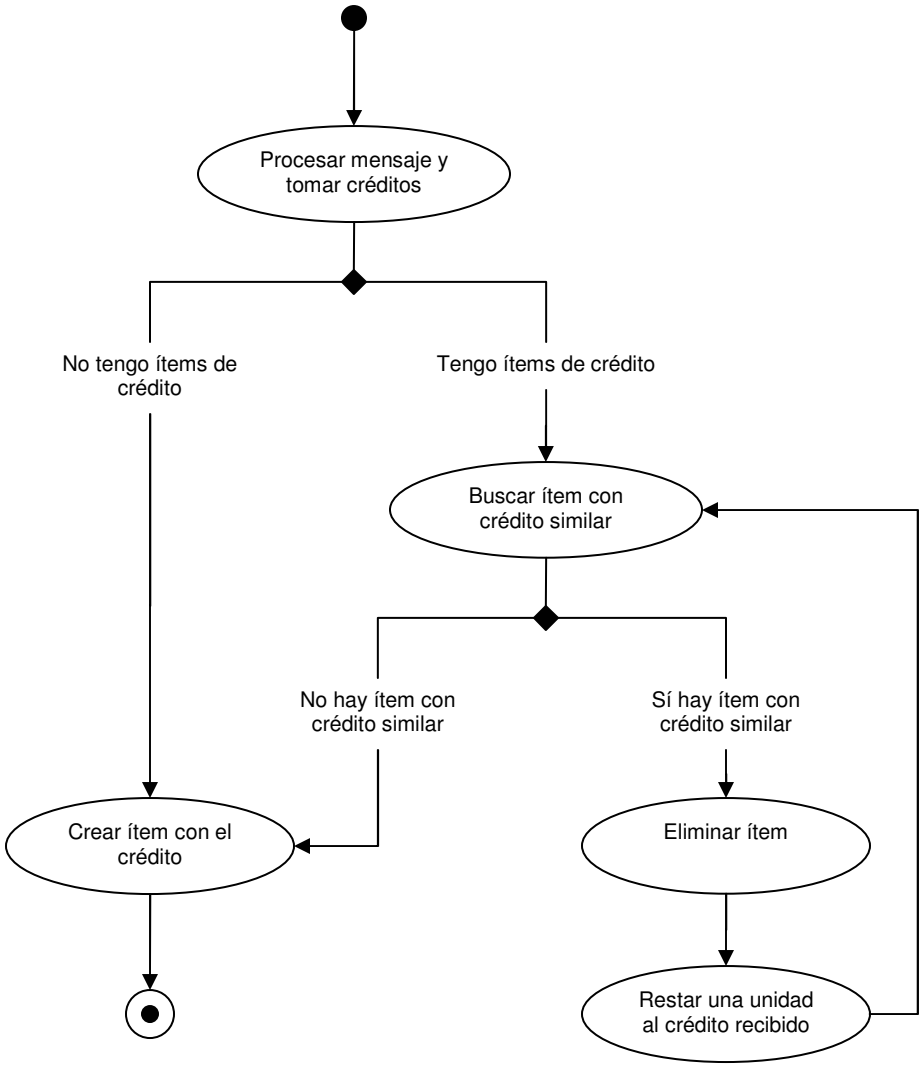


Figura 14.- Al recibir un mensaje con créditos.

2.7.3.3- MÉTODOS BASADOS EN ESPERA DE RESPUESTA

Las esperas de cada procesador por mensajes foráneos son árboles lógicos, por tanto, la detección de terminación también la resolvemos creando un árbol principal de espera de respuesta a partir de la difusión de un mensaje especial de activación. La activación de un procesador no es respondida a su activador hasta que todos sus procesos locales hayan finalizado, y ningún proceso local finaliza hasta recibir respuesta de todos los procesos foráneos finalizados. De esta forma se

imposibilita que no existan procesos en cola mientras hayan mensajes en tránsito, por lo que la detección es eficaz.

Para lograr esto, primero es requerido repartir el mensaje de activación inicial. Al recibir un mensaje de activación inicial:

- Si es la primera vez que se recibe, se guarda la identidad del nodo vecino activador y se propaga la activación a los demás vecinos.
- Si no es la primera vez que se recibe, se rechaza el mensaje mandando un mensaje de rechazo al vecino remitente, de modo que no la espere más tiempo.

Con esto se crea un árbol de expansión en todo el sistema, donde cada nodo tiene un vecino padre-activador.

Una vez que un nodo haya finalizado todos los procesos de su cola:

- Si no ha recibido ningún mensaje de activación inicial entonces debe esperar a recibir uno y luego propagarlo a sus vecinos.
- Si ya recibió la activación inicial, debe esperar a que sus vecinos hijos-activados devuelvan el mensaje de activación inicial y luego devolver su propia activación a su vecino padre-activador.

Cuando el nodo coordinador, que es la raíz del árbol de expansión, recibe de vuelta todos los mensajes de activación iniciales que envió, decreta la pasividad de todos los nodos.

Una ahorro sustancial en el método es no responder cada finalización de proceso disparado, sino contar las finalizaciones. Para evitar saturar el medio de mensajes de reconocimiento, acuses de recibo o mensajes de finalización, cada par de nodos que intercambia mensajes realiza una ***contabilidad de esperas***, es decir, acumulan los mensajes de respuesta que debieron enviarse y sólo realizan un envío de reporte al finalizar sus cargas de trabajo, aligerando drásticamente el peso del algoritmo.

Ahora bien, sabemos que cada proceso disparado que no devuelve respuesta debe ser contabilizado de acuerdo a este enfoque, dado que esas son las esperas críticas sobre las cuales se sostiene la detección, por lo que al momento de detectarse la terminación general, en cada nodo el contador de procesos disparados es igual al contador de respuestas reportadas por sus vecinos. A

partir de esto podemos hacer una variante del algoritmo si utilizamos la diferencia general de ambos contadores como un indicador de terminación en todo el sistema.

Esta variante mejora el desempeño de cada nodo pues no requiere mayor almacenamiento de datos ni mantenimiento de estado al sustituir la espera de respuesta por un contador de esperas que no genera reportes. El reporte final sólo se envía al nodo coordinador al finalizar las actividades y sólo contienen la diferencia local. El coordinador espera los reportes y calcula la diferencia global y espera las respuestas de los mensajes de activación. Las respuestas de activación indican una finalización general y la diferencia nula indica que no quedan mensajes en tránsito.

Las actividades a realizar por cada nodo ante los eventos comunes son:

#1	Al culminar un proceso local y propio.	Nada.
#2	Al culminar un proceso local disparado por un proceso foráneo.	(a) Incrementar el contador de procesos consumidos: `numProcConsumed += 1`
#3	Al disparar un proceso foráneo que me retornará respuesta.	Nada, sólo esperar su respuesta.
#4	Al disparar un proceso foráneo que no me retornará respuesta.	(a) Incrementar el contador de procesos disparados: `numProcLaunched += 1`
#5	Al enviar un mensaje de activación inicial (Dispara un proceso foráneo en el receptor)	(a) Incrementar el contador de procesos disparados: `numProcLaunched += 1`
#6	Al tratar un proceso de activación inicial disparado por un proceso foráneo (Sólo nodo no coordinadores)	(a) Incrementar el contador de procesos consumidos: `numProcConsumed += 1` (b) Si es la primera vez que se procesa este tipo de actividad: (b.1) Anotar el ID del nodo disparador. A este nodo se enviarán los reportes. (b.2) Enviar un mensaje de activación inicial a cada vecino (incurre en *n* actividades #5)
#7	Al vaciarse la cola de procesos.	(a) Calcular la diferencia entre los procesos disparados y los procesos consumidos localmente:

		`Report = numProcLaunched - numProcConsumed` (b) Enviar un mensaje de reporte al nodo vecino disparador.
#8	Al recibir un mensaje de reporte.	(a) Sumar al contador de procesos disparados el valor del reporte: `numProcLaunched += ReportReceived`

2.8.- COMPARACIÓN DE LA IMPLEMENTACIÓN DE LOS MÉTODOS

2.8.1.- Mensajes de activación y retardo para empezar a trabajar

Métodos basados en créditos

Los nodos pueden empezar a trabajar antes de la repartición de créditos, sin embargo no pueden enviar mensajes sin ellos, por lo que la activación de cada procesador está supeditado al hecho de tener créditos a menos que éstos puedan deberse. Los mensajes de activación inicial pueden entregarse sin mayor control, puede haber una difusión de créditos sin necesidad de rechazar mensajes de vecinos. El campo necesario en el mensaje es un valor real o un valor entero según sea el caso.

El peor escenario en la repartición inicial es que el primer nodo reparta créditos y luego todos los demás reparten créditos a todos sus vecinos, ya sea esto por demasiada rapidez o muy poco retardo ninguno entra en cuenta con suficiente tiempo que va a enviar créditos a nodos de quienes reciben créditos también, inundando los canales con mensajes innecesarios e incurriendo en procesamiento adicional. En todo caso sólo sería dos mensajes por cada par de nodos.

En una topología completa de *n* nodos la cantidad total de mensajes sería:

n-1 mensajes del coordinador + n-2 mensajes de cada nodo = (n-1) + (n-2)(n-1)*

En el mejor caso de una topología completa, el coordinador reparte créditos a todos, luego sólo uno de los vecinos reparte, luego otro vecino y así sucesivamente de modo que entre ningún par viajan más de un mensaje de activación. El total de mensajes sería:

*n-1 + n-2 + n-3 + … + n-n = n*n – (1 + 2 +3 + … + n)*

De igual manera, a tal situación utópica tampoco corresponde mucho provecho pues el tiempo que no están activados tampoco pueden ocupar el canal por no tener créditos.

Métodos de espera de respuesta

Son necesarios los mensajes de activación pero los procesadores pueden empezar a trabajar sin ellos y pueden enviar mensajes. Cada nodo puede inclusive terminar todas sus tareas sin recibir mensajes de activación, los cuales son necesarios sólo para crear el árbol de expansión. Los mensajes de activación no tienen campos adicionales además del tipo de mensaje: Activación, Respuesta o Rechazo (una respuesta temprana). Si los procesadores trabajan a la par en cuanto a velocidad, el número de mensajes duplica a los métodos de créditos, puesto que a cualquier mensaje de activación adicional se le replica un mensaje de rechazo para evitar ciclos en el árbol de expansión. Así, el peor caso en una topología completa de *n* nodos: Cada procesador excepto el coordinador envía *n-2* mensajes de activación (excluye al coordinador y a sí mismo), y rechaza *n-2* mensajes (no rechaza al coordinador ni recibe de sí mismo).

n-1 mensajes de coordinador + (n-2)(n-1) mensajes de activación + (n-2)*(n-1) rechazos = (n-1) + 2*(n-2)*(n-1)*

El caso utópico en la topología completa sería una propagación escalonada por retraso, por ejemplo, un canal compartido y procesadores con cierto retardo, de modo que un primer nodo envía n-1 mensajes, el segundo recibe de primero, procesa y envía n-2 mensajes, el tercero recibe mensajes del primero y del segundo y consecuentemente envía n-3 mensajes (excluye al primer nodo, al segundo y a sí mismo), el cuarto procesa tardíamente y envía n-4 y así sucesivamente. Por tanto:

El coordinador envía n-1 mensajes, no recibe rechazos +
el segundo envía n-2 mensajes y n-2 rechazos +

el i-ésimo envía n-i mensajes y n-i rechazos = (n-1) + 2(n-2) + 2*(n-3) + … + 2*(n-n) mensajes*

Métodos de Propagación de Consulta

La cantidad de datos que comparten es mínima y no requieren mensajes iniciales de control. Los nodos pueden empezar a trabajar inmediatamente.

2.8.2.- Almacenamiento de datos

Métodos basados en créditos

Utilizando valores reales, se requiere que cada nodo contenga un campo en punto flotante que bien podría ser *Float* de 32 bits o *Double* de 64 bits para mejorar la precisión. Otro valor a almacenar es la identificación de algún vecino a quien deban entregar los créditos para retornarlos al nodo detector, 32 bits podrían ser suficientes, como por ejemplo, un dirección IP.

Si se usa el logaritmo negativo binario de los créditos sólo se requieren valores enteros de 32 bits o menos, sin embargo mantener una lista de créditos necesita más espacio, un ítem de crédito requiere un campo de valor y un apuntador al siguiente crédito, a menos que se utilice un arreglo de tamaño variable.

Métodos de espera de respuesta

El proceso de repartición de mensajes de activación y la creación del árbol de expansión requiere que cada nodo almacene información de sus nodos hijos y su padre: *32 bits * la cantidad de vecinos* en el peor de los casos. Adicionalmente un dato con información de devolución del mensaje de activación: *Cantidad de vecinos * 1 booleano*. La contabilidad de esperas de respuestas requiere en el peor de los casos un valor entero por cada vecino: *Cantidad de vecinos * 32 bits* por ejemplo en el caso de arreglos. Una lista dinámica requiere mayor espacio.

Contabilizando *procesos disparados - procesos consumidos* la información es menor: Un valor entero.

Métodos de propagación de consulta

Cada nodo debe almacenar como mínimo información acerca de:

- Si envió o no mensajes de actividad después del último ciclo de consulta, por ejemplo un valor del tipo booleano. El uso de banderas de 1 bit requiere operaciones lógicas adicionales.
- Si inició o no el ciclo de consulta actual, 1 booleano.
- Una lista de los vecinos consultados en el ciclo. A menos que se utilice un arreglo, un ítem de lista tiene mínimo un campo de valor y un campo de referencia al siguiente elemento.
- La identidad/dirección del vecino a quien se debe responder el ciclo de consulta, 32 bits.

2.9.- PRUEBAS Y ANÁLISIS

Se han presentado varios esquemas para la detección de terminación con características distintas y similares, que si bien no son las únicas, cubren parcialmente bien la mayoría de las técnicas propuestas por distintos autores, tratando de ser flexibles respecto a topologías de comunicación y dinámica de los sistemas, es decir, los métodos que hemos descrito no están encadenados a una topología fija, permitiendo el cambio en el número de nodos y la creación dinámica de procesos.

El método ideal es aquél que permita la más rápida detección con un mínimo de interferencia en el sistema de aplicaciones, además de ser fácilmente implementable y no imponga demasiadas restricciones. Una característica deseada sería que pudiese ser asimilada en un sistema ya existente y funcional sin demasiado esfuerzo. Obviamente no existe un método que cumpla con todas estas características, pero al momento de escoger debe seleccionarse la que mejor se acomode a las necesidades.

A partir de las implementaciones podemos decir que:

1. Algunos afectan directamente la forma de trabajo del sistema, como por ejemplo los *métodos de crédito*, que implican construir mensajes con campos adicionales así como añadir un retardo al comienzo también.

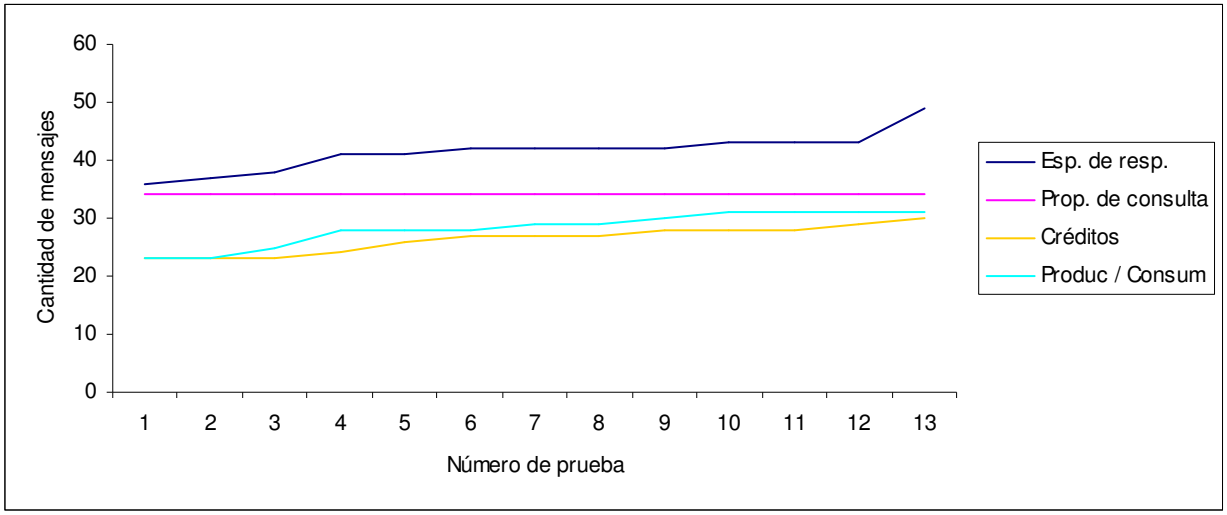
2. El número de mensajes en cada método es una cifra variable afectada por la topología de la red y el orden de los nombres en las listas de vecinos, no sólo por los eventos del sistema y las actividades trabajadas.
3. Mientras que el sistema de comunicación se parezca más a un sistema monoprocesador con buses de comunicación compartidos, el caudal de mensajes y el tamaño de éstos afectará más.
4. Las técnicas no son a prueba de caídas y mientras menos información compartan los nodos y menos datos del sistema en general tenga cada uno, el método es más suceptible a las fallas de la red. Para soportar caídas parciales el sistema debe conservar cierto mínimo de información e intercambiar más mensajes entre nodos.

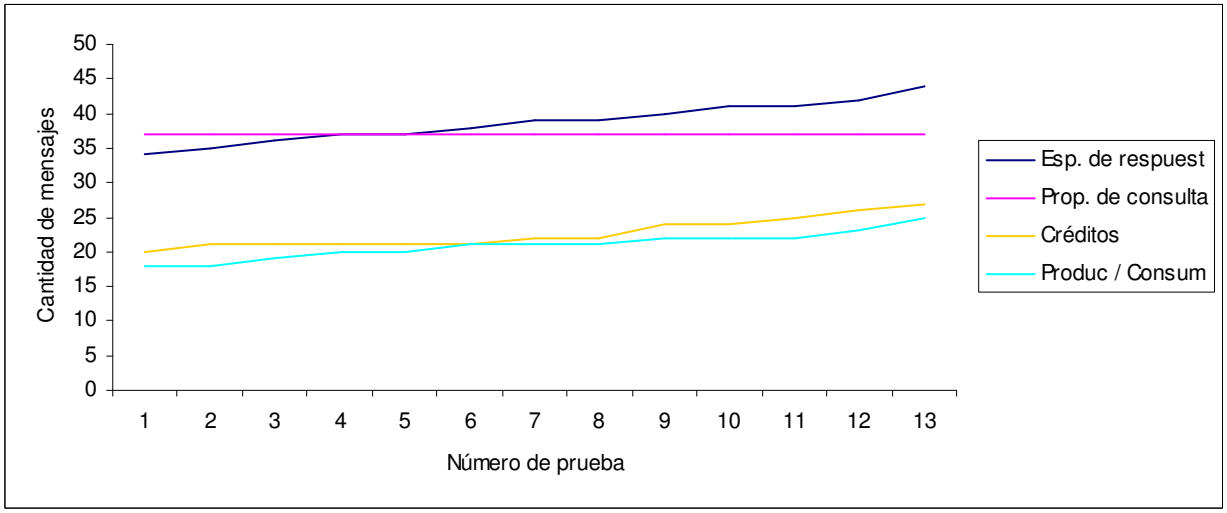
Los *métodos de crédito* requieren menos mensajes especiales, la única desventaja es que los mensajes quedan encadenados al método al necesitar empaquetarlos con un campo de créditos, y en términos de desarrollo, significa alterar el módulo de mensajería o quizás modificar funciones ya implementadas. También es necesario impedir que los procesos envíen mensajes mientras sus localidades no tengan créditos, lo cual en sistemas ya existentes puede ser una modificación en niveles bajos de software no sencilla.

Otra opción para pocos mensajes es la *contabilidad de procesos disparados y consumidos*, con operaciones mucho más sencillas que el mantenimiento de créditos y sin listas de elementos. La cantidad de mensajes en este método pueden minimizarme ligeramente más si los mensajes de activación excedentes que se reparten al crear el árbol de expansión no se rechazan, por lo que los nodos padres no esperan a los hijos para notificar sus propias finalizaciones y supeditando la detección a la diferencia de procesos disparados y consumidos únicamente. Las pruebas realizadas en topologías distintas y con diferentes coordinadores demuestran un consumo menor de mensajes (**Figuras 17** y **20**). En realidad aquí los mensajes de iniciación no necesitan entregarse de manera controlada, como rechazar mensajes excedentes y acusar al finalizar. Sabemos que

(i) en un intervalo de actividad (cualquier período de trabajo-descanso) a *n* procesos locales disparados le corresponden un (1) reporte de `Disparados-Consumidos`, y además,

(ii) el primer reporte acusa siempre el mensaje de activación inicial.





Figuras 17.- Mensajes de control transmitidos en una topología con ciclos (Anexos, Prueba I, Tablas 1, 3, 5 y 7).

Como es lógico, cualquier diferencia nula puede producir un falso positivo en la detección, por lo que la técnica es eficaz sólo a medida que el número de procesos por procesador es elevado y se reducen las probabilidades para una falsa resta culminante. A partir de (ii) sabemos que los mensajes de activación sólo sirven para crear un camino lógico hasta el nodo detector de terminación pero no resuelven los falsos positivos, sin embargo, si se le obliga a los nodos padres no enviar su primer reporte hasta recibir el primer reporte de todos sus hijos, las probabilidades de una diferencia 0 (condición de terminación) se anulan en buena parte. (i) nos apunta que aunque hayan mensajes de activación controlados, el número de mensajes requerido siempre estará acotado por debajo de los necesitados en una *contabilidad de esperas de respuesta*.

La cantidad de mensajes es una forma de evaluar el peso del método sobre el sistema, sin embargo, lo importante no es el total de mensajes sino el número de mensajes por cada canal de comunicación y más aún, la forma en que interfieren los mensajes en las operaciones ordinarias de los nodos involucrados. En un sistema con canal de comunicación compartido por todos, cada mensaje tiene un peso mucho mayor que en un sistema difuso con varios canales. A pesar que las *propagaciones de consulta* requieren mayor número de mensajes para la detección de pasividad general sus operaciones y mensajes no necesitan mayor atención durante los intervalos de actividades ordinarias, y la demanda de procesamiento es realmente bajo.

El número de mensajes también tiene relación con el número enlaces en la red distribuida y su topología, y esta dependencia varía entre los métodos. En las *propagaciones de consulta*, la escogencia de un nodo u otro para iniciar las labores de detección afecta el número de mensajes totales. La **Figura 18** muestra el total de mensajes utilizados al iniciar cada ciclo de consulta a partir de un nodo distinto en dos topologías. En todas las pruebas todos los nodos enviaron mensajes, y como los ciclos de consulta no avanzan en nodos trabajando, la variación entre una prueba y otra no restaña la similitud entre los resultados. Conociendo la forma en que trabaja un sistema es posible escoger un nodo inicial ideal, sin embargo, a esto hay que añadirle unas tablas de vecinos específicas también, pues el orden de entrega es otro factor influyente. El grafo más favorable para la *propagación de consulta* es el anillo (**Figura 19**).

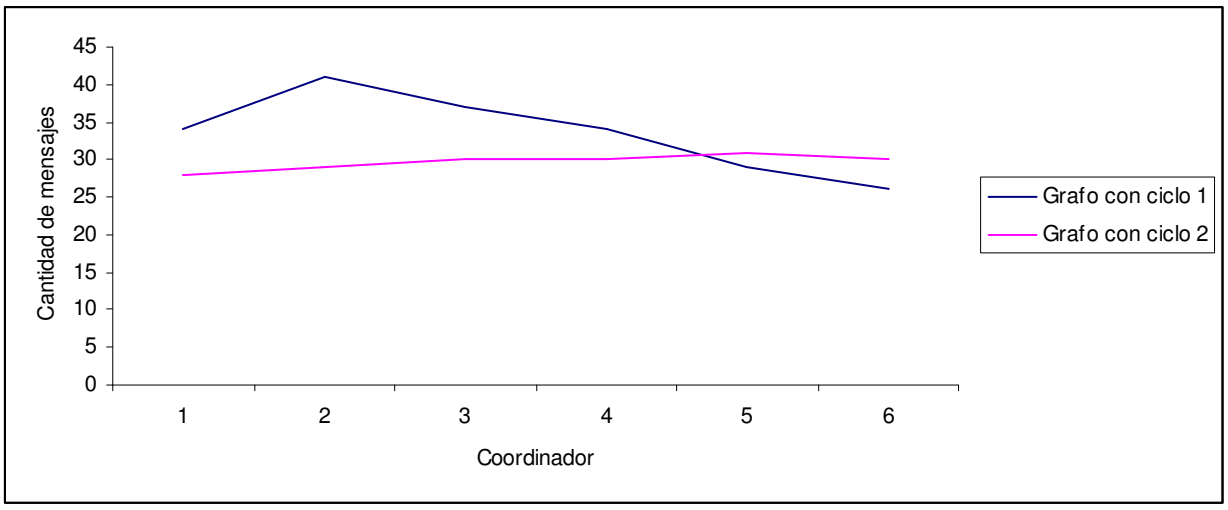


Figura 18.- Mensajes de control al propagar consultas (Anexos, Prueba II, Tabla 9).

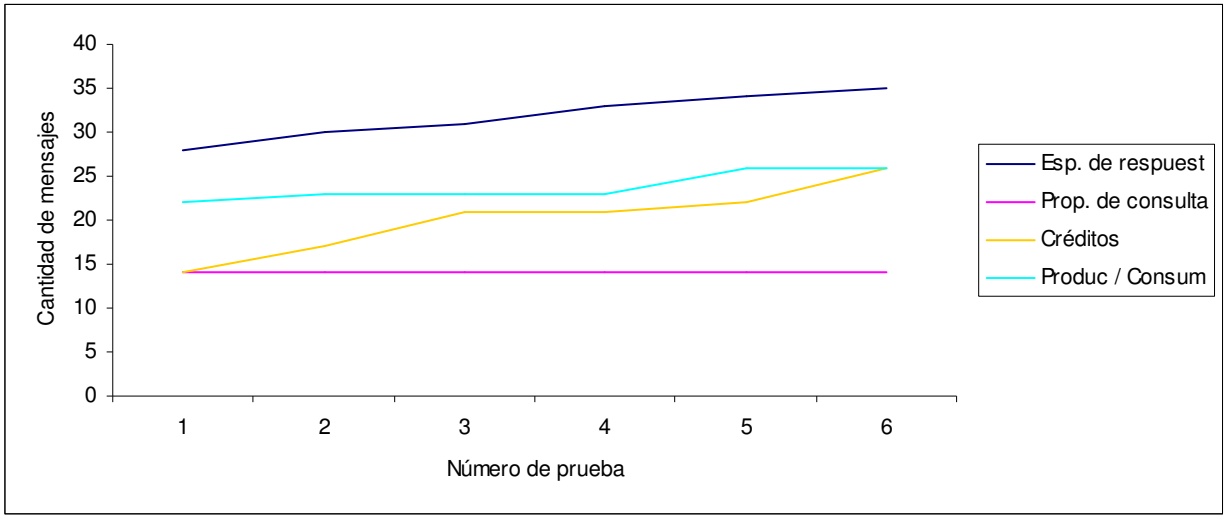
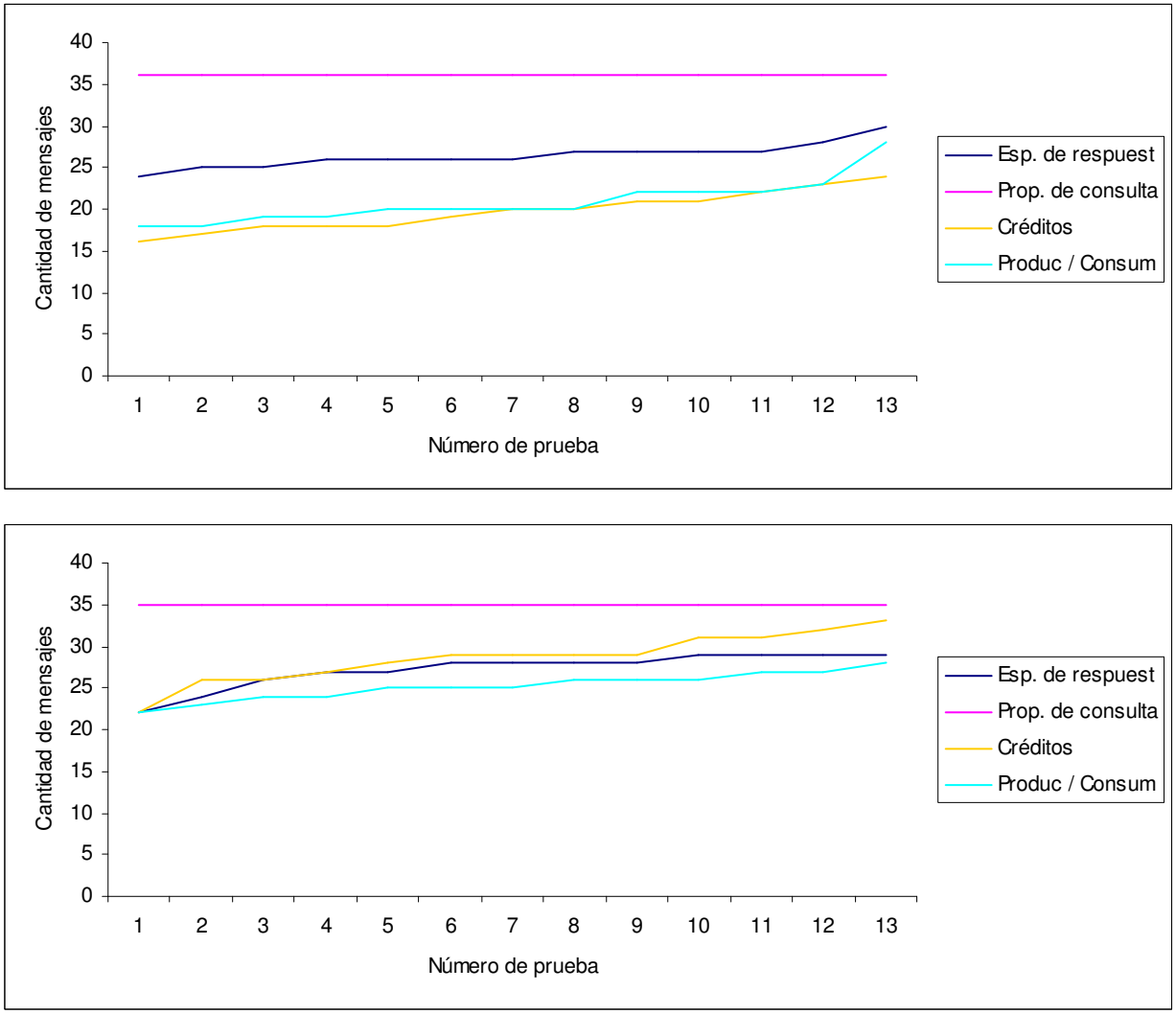


Figura 19.- Mensajes de control transmitidos en una topología de anillo (Anexos, Prueba III, Tabla 10).

Ahora bien, la *espera de respuesta* está más afectada por la cantidad de enlaces que hayan en la red, pues como ya se ha dicho, a cada proceso disparado sin respuesta le corresponde un reporte. Por eso las esperas de respuesta requieren mayor número de mensajes en grafos completos o con ciclos, eso lo demuestra el promedio de mensajes requeridos entre distintas ejecuciones sobre la topología con ciclo en las **Figuras 17**, el cual es superior al promedio utilizado con propagaciones de consulta. Sin embargo en árboles tiene un mejor desempeño que las consultas, compare el comportamiento del método entre las **Figuras 17** y **20**.

En términos reales el número de mensajes no varía mucho entre un método y otro, pues una diferencia de unos cuantos mensajes no hacen mayor diferencia a altas velocidades de comunicación, ni tampoco unos cuantos cientos de operaciones adicionales afectarán sensiblemente el desempeño del sistema distribuido en general. Sin embargo hay que recordar que en un sistema distribuido no solamente hacen falta algoritmos de detección de terminación, sino que son precisos otros algoritmos de control importantes para el funcionamiento. Escoger entre un método y otro está influenciado por la facilidad que tienen en incorporarse al sistema en particular y al grado de flexibilidad para permitir el trabajo normal de los otros controles necesarios. En este sentido el método de créditos es el menos flexible para cualquier sistema y además consume mayor cantidad de operaciones y recursos, mientras que los ciclos de consulta requieren más mensajes y tiempo para detectar terminaciones pero son más fáciles de adaptar a cualquier situación.



Figuras 20.- Mensajes de control transmitidos en una topología de árbol (Anexos, Prueba I, Tablas 2, 4, 6 y 8).

Sin embargo, la cantidad de espacio utilizada en memoria por cada programación tampoco es realmente muy significativa, una inspección al bloque de código de nuestras implementaciones nos muestra que el espacio requerido para los datos es apenas una mínima porción del total necesitado para cargar la implementación. Podemos comparar el espacio de datos que consume cada uno:

Cantidad de espacio mínimo requerido para las actividades de carga/consumo de procesos y envío/recepción y procesamiento de mensajes en la capa de aplicación (datos elementales)

Cantidad fija utilizada en datos de control	17 enteros 4 apuntadores
Cantidad por cada vecino	1 entero 1 booleano
Cantidad por cada proceso en memoria	6 enteros 3 apuntadores
Cantidad por cada mensaje en espera	9 enteros 1 apuntador
Cantidad por cada espera de proceso foráneo	1 entero 1 apuntador

Cantidad extra requerida en el bloque de datos de cada método

	Créditos	*Espera de Resp.*	*Produc / Consum*	*Prop. de consulta*
Cantidad fija utilizada en datos de control	7 enteros 1 booleano 1 apuntador	8 enteros 3 booleanos	6 enteros 1 booleano	5 enteros 3 booleanos
Cantidad variable utilizada en datos de control	*Por ítem de crédito:* 1 entero 2 apuntadores			
Cantidad por cada vecino		2 enteros 1 booleano	1 booleano	1 booleano
Cantidad por cada mensaje en espera	1 entero	1 entero	1 entero	

2.10.- EVALUACIÓN DE USABILIDAD Y DIDÁCTICA DE LA APLICACIÓN

Para evaluar de manera empírica el instrumento, un conjunto de usuarios manejó la aplicación previamente a responder una serie de preguntas relacionadas con la usabilidad, didáctica y ejecución del software. A los usuarios se les introdujo cada algoritmo de detección de terminación y luego se le presentó la animación de cada método. El formulario de preguntas fue:

La aplicación que usted utilizará está dispuesto para circunstancias didácticas. Después de utilizar el software trate de responder las preguntas que se formulan a continuación. A partir de sus respuestas evaluaremos la aplicación con el propósito de mejorar su diseño.

1) ¿Utilizó con facilidad el software?

2) ¿Cuáles algoritmos visualizó usted?

3) ¿Conocía usted previamente alguno de los algoritmos animados?

4) ¿Comprendió las animaciones? ¿Comprendió los algoritmos simulados? ¿Fueron las animaciones suficientemente claras mostrando la idea de los algoritmos?

5) ¿Le parecieron triviales los algoritmos animados?

6) ¿Cree que es necesario conocer previamente los algoritmos para comprender las animaciones?

7) ¿Terminaron las animaciones iniciadas? ¿Finalizaron todos los procesos al decretarse la finalización general del sistema?

8) ¿Tardaron mucho las animaciones para finalizar?

9) ¿Configuró usted la animación? ¿Probó alterar alguna de las variables configurables de la animación?

10) ¿Probó algún algoritmo más de una vez? De ser así ¿Notó diferencias entre los distintos intentos?

11) ¿Quedó usted satisfecho con la estocástica del sistema?

12) ¿Recomienda usted agregarle algo a la aplicación? ¿Cambiaría o quitaría algo a las animaciones?

13) ¿Cree que los elementos de animación distraen la atención? ¿Cree que la interfaz de animación desvirtúa el propósito del software?

14) ¿En algún momento le pareció tediosa la animación?

Podemos abstraer las preguntas consultadas en tres reglones:

I) Las preguntas 3, 4, 5, 6, 8 y 12 están relacionadas con el propósito didáctico de la aplicación.

¿Conocía usted previamente alguno de los algoritmos animados?

No	Algunos
66,6 %	33,3 %

¿Comprendió las animaciones? ¿Comprendió los algoritmos simulados? ¿Fueron las animaciones suficientemente claras mostrando la idea de los algoritmos?

Sí 100 %

¿Le parecieron triviales los algoritmos animados?

No 66,6 %	Algunos 33,3 %

¿Cree que es necesario conocer previamente los algoritmos para comprender las animaciones?

Sí 100 %

¿Tardaron mucho las animaciones para finalizar?

No 16,6 %	Un poco / Regular / A veces 50 %	Sí 33,3 %

¿Recomienda usted agregarle algo a la aplicación? ¿Cambiaría o quitaría algo a las animaciones?

Nada 33,3 %	Sí (Ver Anexos) 66,6 %

II) Las preguntas 7, 9, 10 y 11 conciernen a la eficacia del software en cumplir con lo que se exige relativo a la simulación de los métodos.

¿Terminaron las animaciones iniciadas? ¿Finalizaron todos los procesos al decretarse la finalización general del sistema?

Sí 100 %

¿Configuró usted la animación? ¿Probó alterar alguna de las variables configurables de la animación?

Sí 100 %

¿Probó algún algoritmo más de una vez? De ser así ¿Notó diferencias entre los distintos intentos?

Sí notó diferencias 66,6 %	Ningún cambio sustancial. 33,3 %

¿Quedó usted satisfecho con la estocástica del sistema?

Sí 100 %

III) Las preguntas 1, 2, 8, 9, 12, 13, y 14 tienen que ver con la usabilidad de la aplicación y su interfaz.

¿Utilizó con facilidad el software?

Sí 100 %

¿Cuáles algoritmos visualizó usted?

Todos 100 %

¿Tardaron mucho las animaciones para finalizar?

No 16,6 %	Un poco / Regular / A veces 50 %	Sí 33,3 %

¿Configuró usted la animación? ¿Probó alterar alguna de las variables configurables de la animación?

Sí 100 %

¿Recomienda usted agregarle algo a la aplicación? ¿Cambiaría o quitaría algo a las animaciones?

Nada 33,3 %	Sí (Ver Anexos) 66,6 %

¿Cree que los elementos de animación distraen la atención? ¿Cree que la interfaz de animación desvirtúa el propósito del software?

No 66,6 %	Quizás la música 33,3 %

¿En algún momento le pareció tediosa la animación?

No 100 %

Los resultados de la encuesta (ver Anexos, Exámen de Aplicación con Usuarios) son en general positivos para la aplicación, con aceptación casi total en cuanto a didáctica, eficacia y usabilidad. Los encuestados concluyen que el software es útil para explicar los algoritmos pero no es didácticamente independiente por sí solo, requiriendo que los usuarios conozcan someramente los métodos previo a visualizar las animaciones. En efecto, la aplicación resultante no busca sustituir la función del tutor

en las aulas de clase mas sí facilitar la exposición de los algoritmos para éste, que es el propósito original.

Tras la aplicación del éxamen se recibieron algunas críticas y recomendaciones comunes, como por ejemplo la carencia de una distinción visual para el proceso coordinador en los métodos, o la necesidad de explicaciones tácitas de algunos mensajes y procedimientos, o bien que eventualmente las simulaciones se extendían mucho tiempo, las cuales fueron aceptadas y se realizaron las modificaciones y mejoras pertinentes en la aplicación.

2.11.- FIGURAS DE LA APLICACIÓN FINAL

Las siguientes figuras muestran las imágenes de la aplicación final resultante.

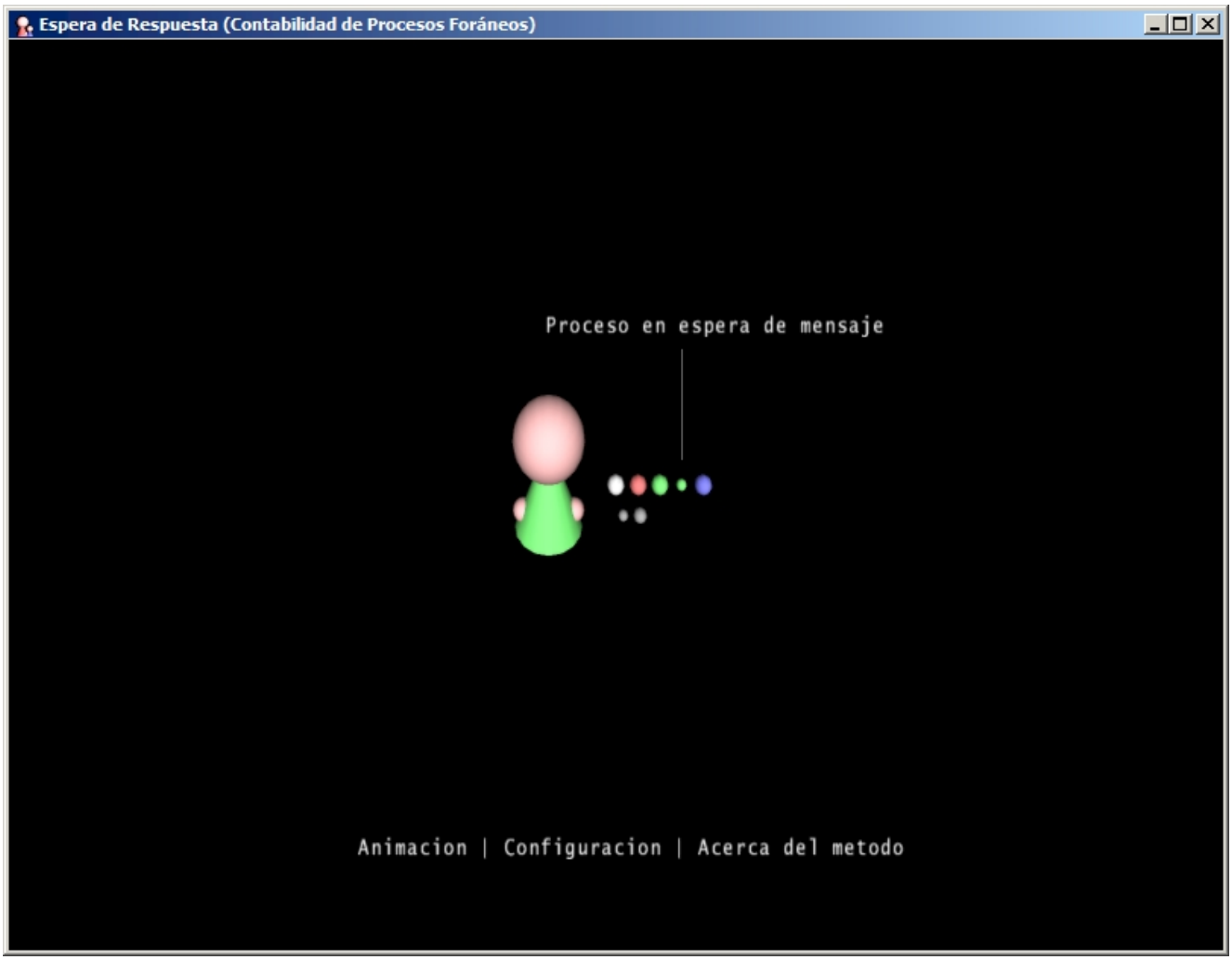


Figura 21.- Imagen de la pantalla inicial de la aplicación.

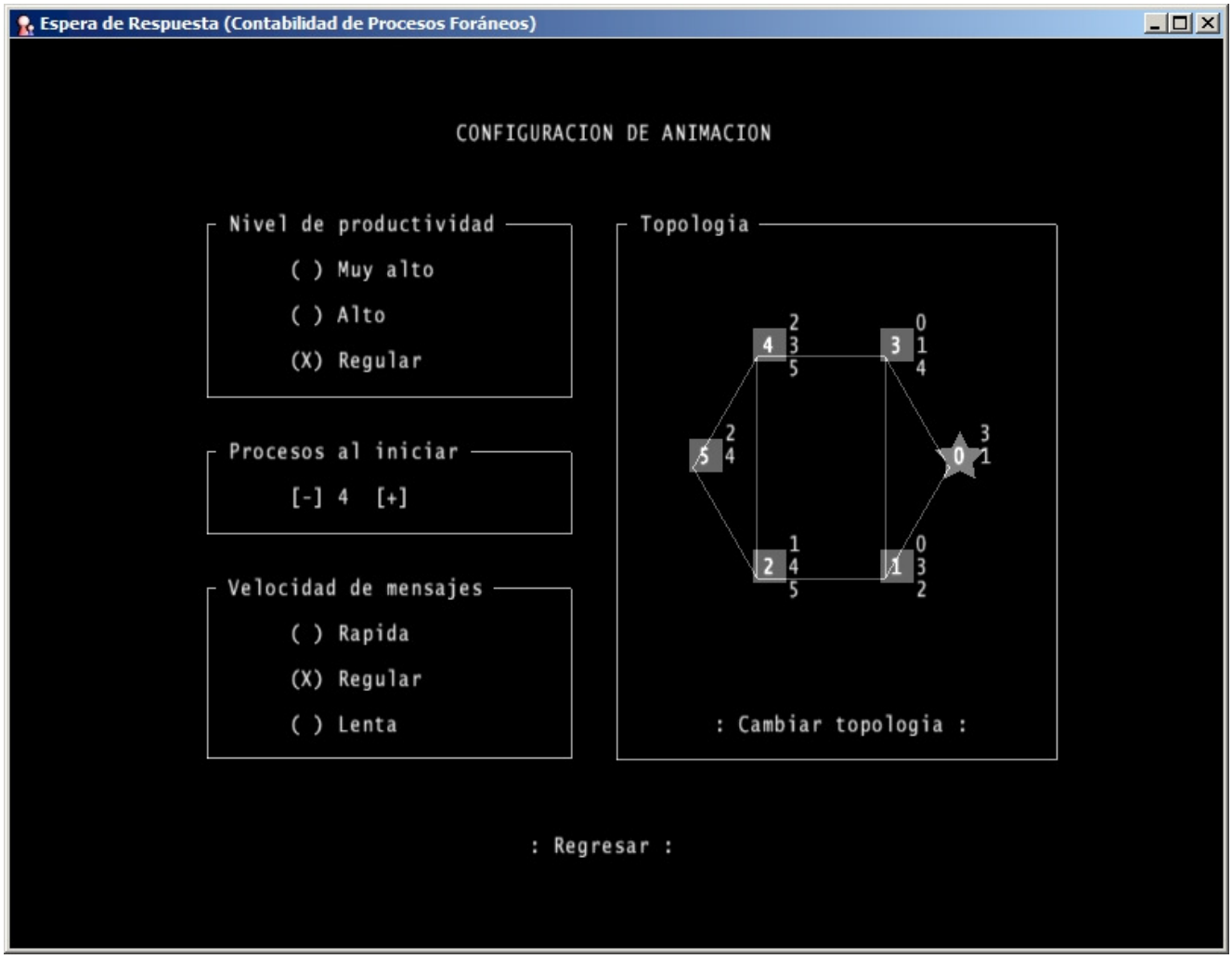


Figura 22.- Imagen de la configuración de una animación.

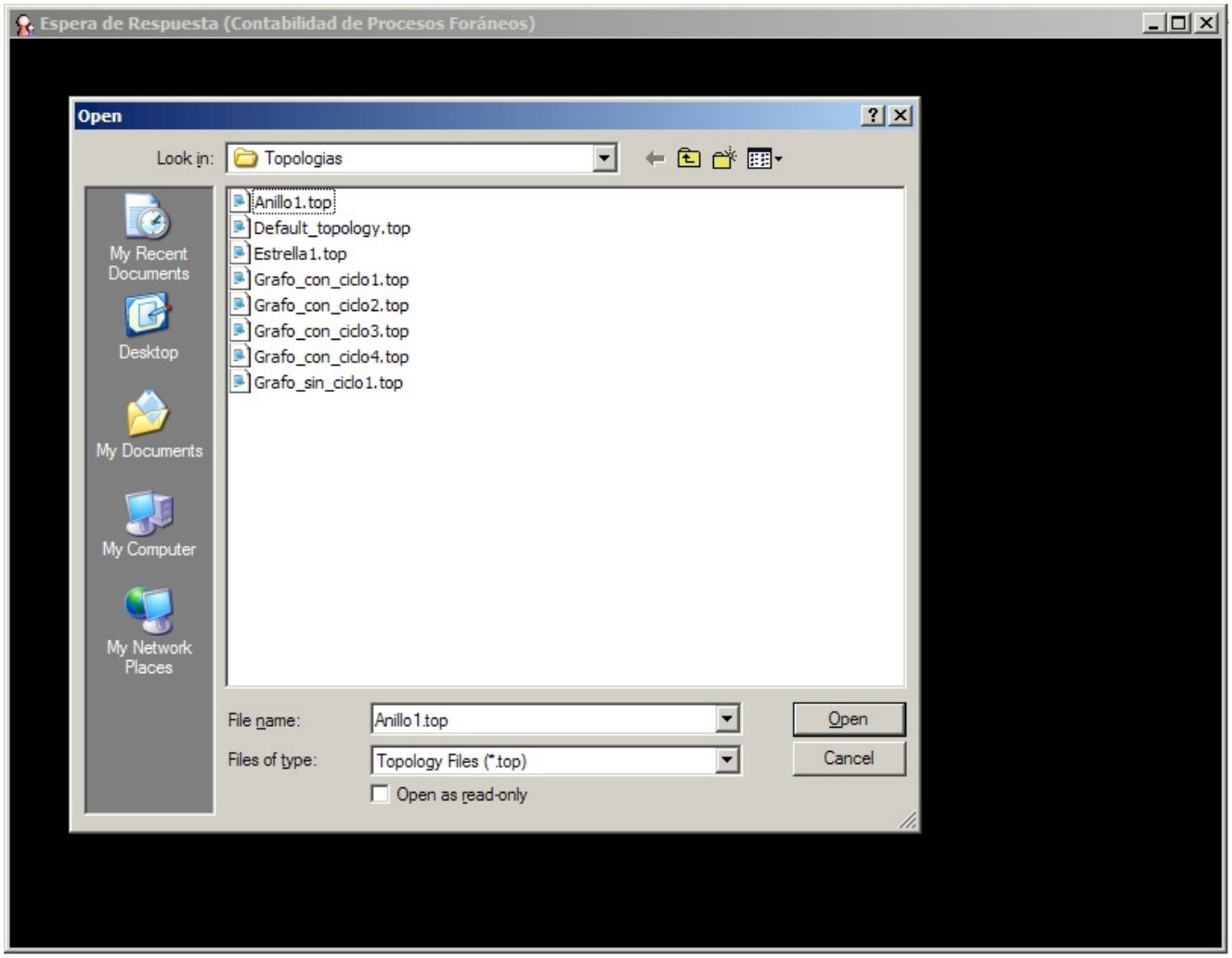


Figura 23.- Ventana de selección de topología

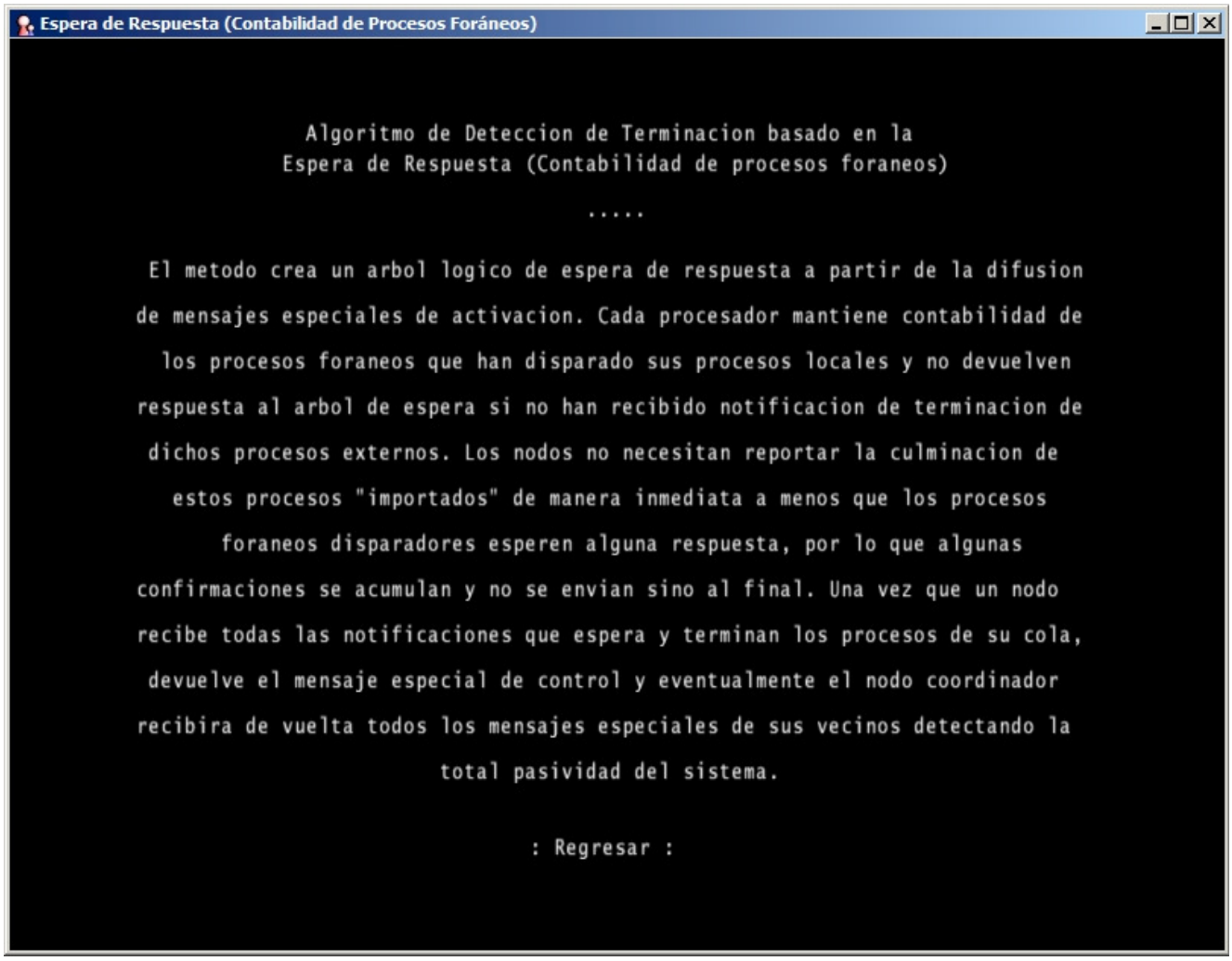


Figura 24.- Imagen del resumen de un método.

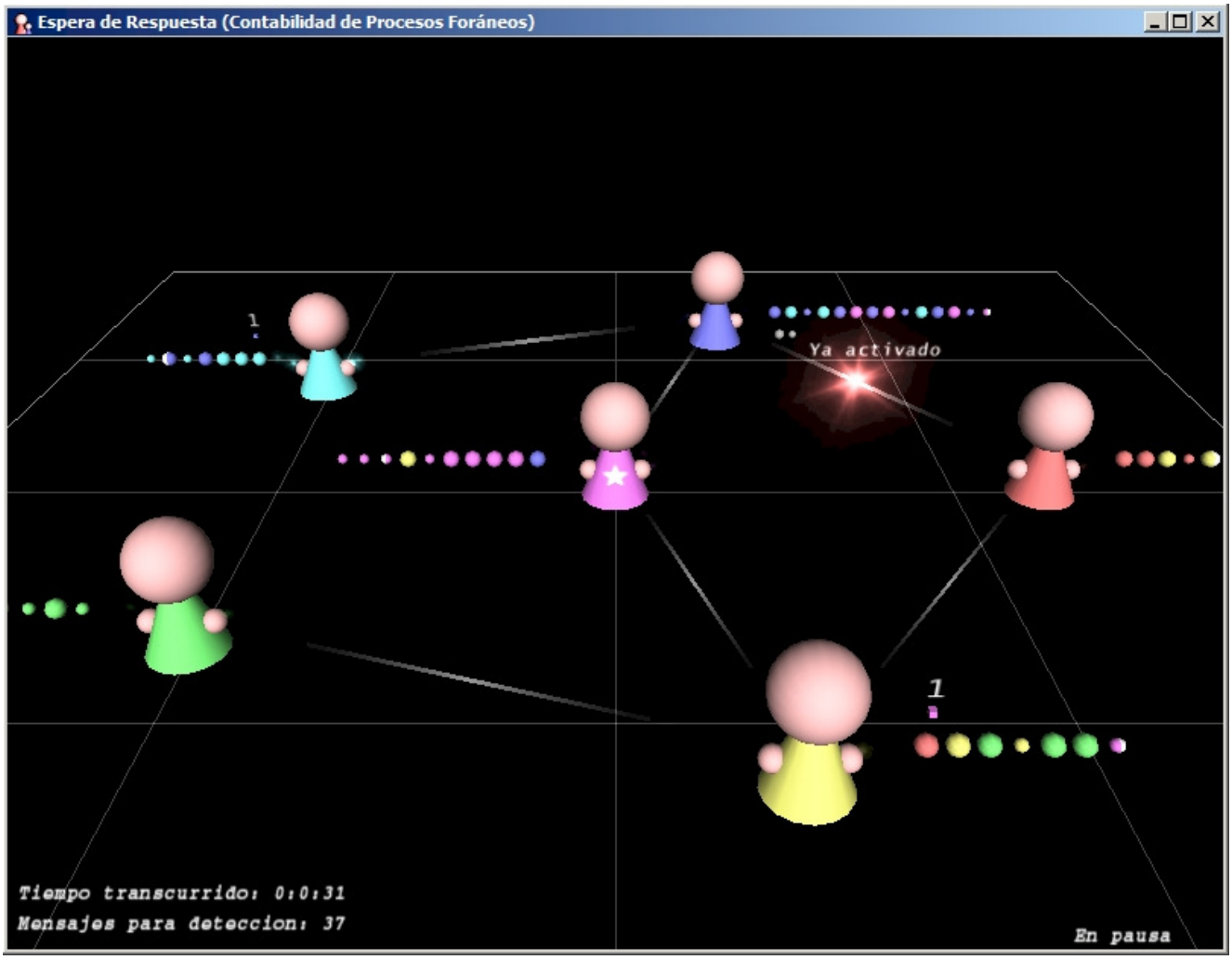


Figura 25.- Imagen de una animación en ejecución.

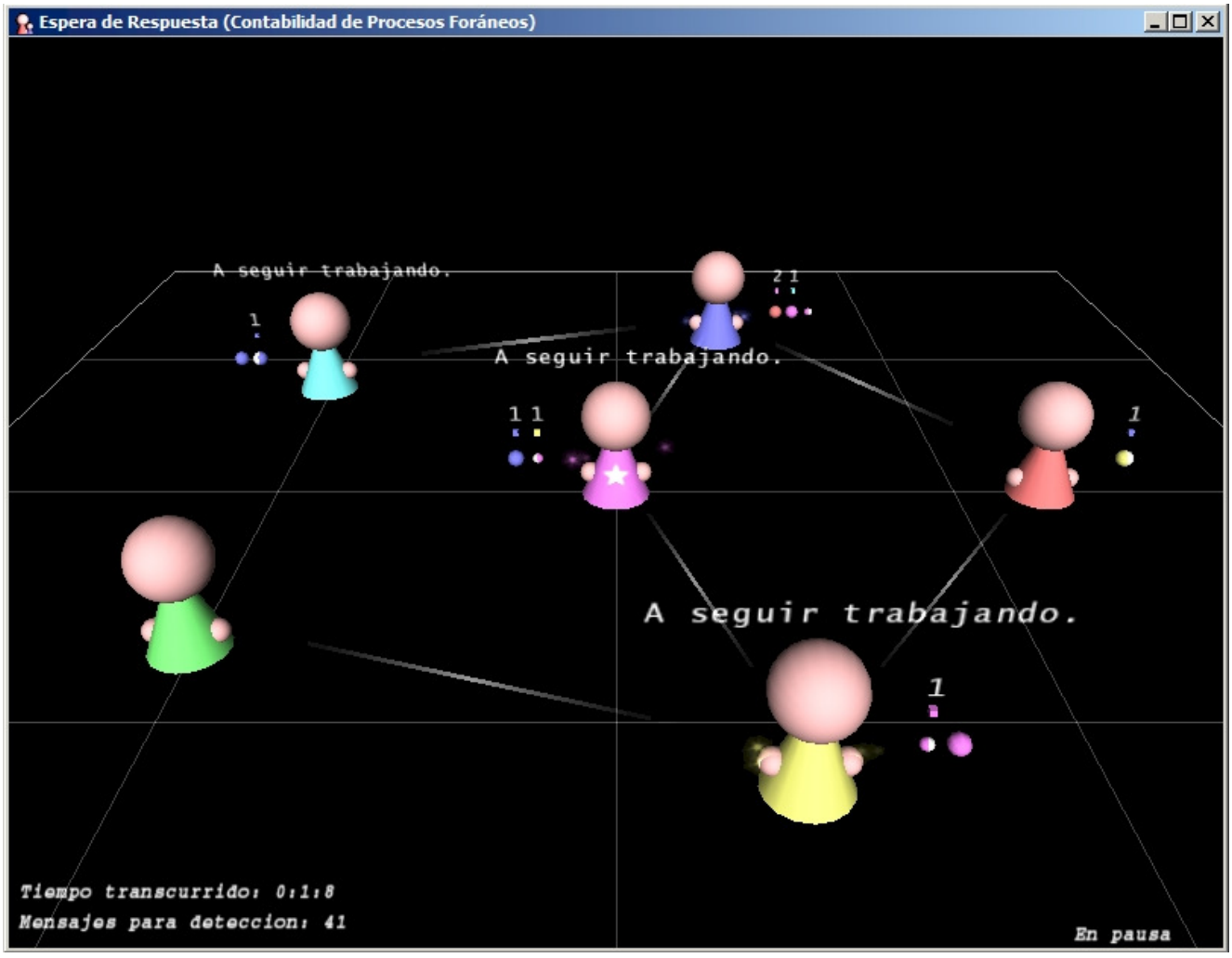


Figura 26.- Imagen de una animación en ejecución.

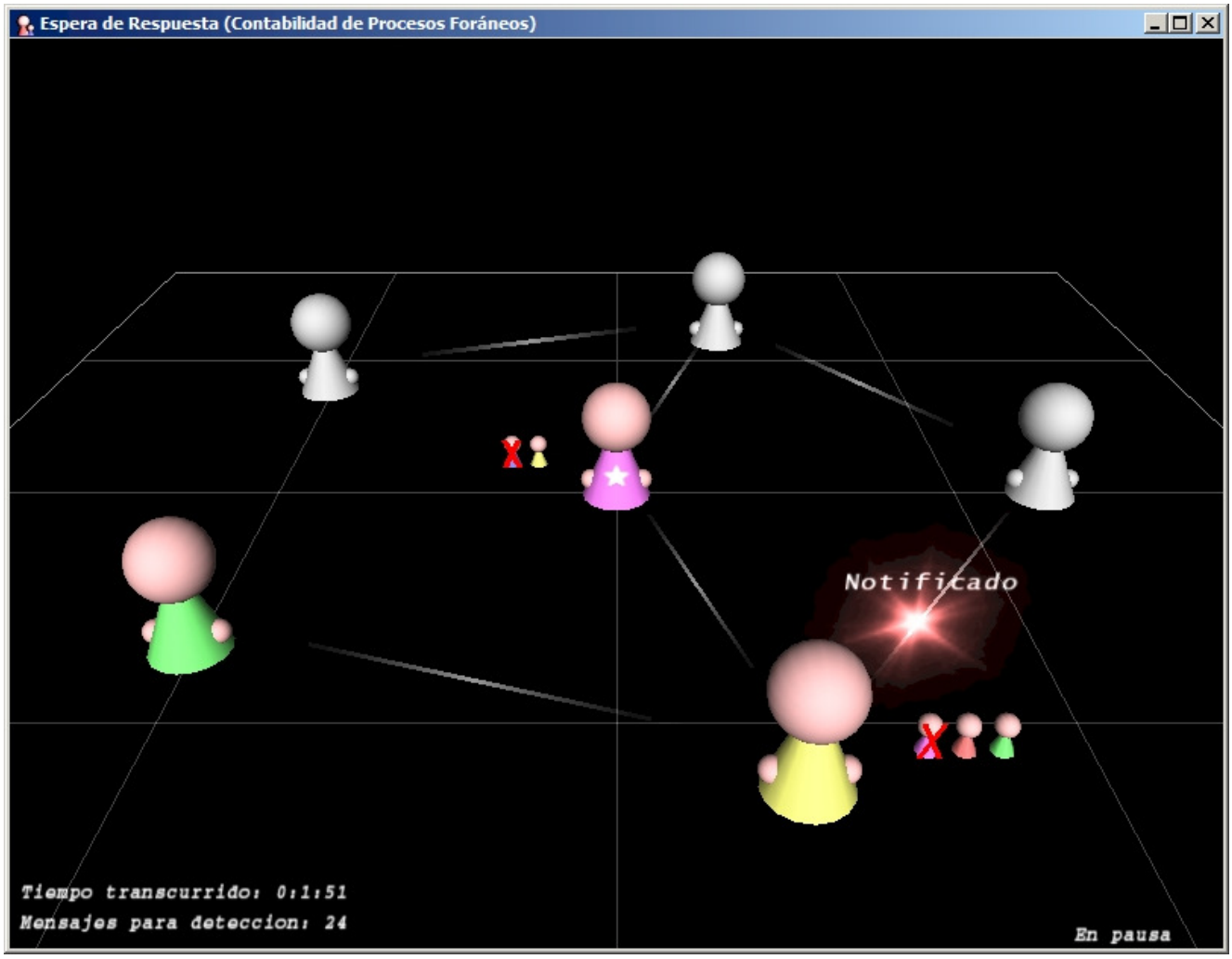


Figura 27.- Imagen de una animación en ejecución.

2.12.- TOPOLOGÍAS Y FORMATO

La aplicación acepta archivos que contengan la topología a representar, las topologías deben seguir el siguiente formato:

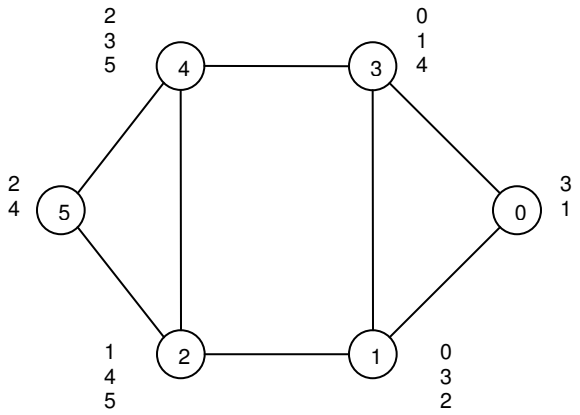
Número de nodos
Coordenadas normalizadas del nodo 0 / Cantidad de vecinos del nodo 0 / Lista de vecinos de 0
Coordenadas normalizadas del nodo 1 / Cantidad de vecinos del nodo 1 / Lista de vecinos de 1
…
…
Coordenadas normalizadas del nodo i / Cantidad de vecinos del nodo i / Lista de vecinos de i

Para la animación las coordenadas deben estar normalizadas en el espacio tridimensional, es decir, dentro del rango [-1, 1] (extremos inclusive) en X, Y y Z. El orden de los nombres en las listas de vecinos especifican el orden en que se enviarán mensajes de difusión u otros mensajes de control.

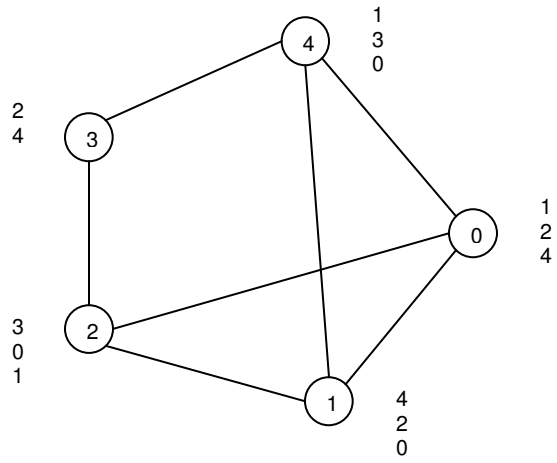
Por ejemplo, las siguientes líneas representan un sistema de 4 nodos con topología de anillo con centro (0,0,0):

```
4
0.5 0.0 0.5 2 3 1
-0.5 0.0 0.5 2 0 2
-0.5 0.0 -0.5 2 1 3
0.5 0.0 -0.5 2 2 0
```

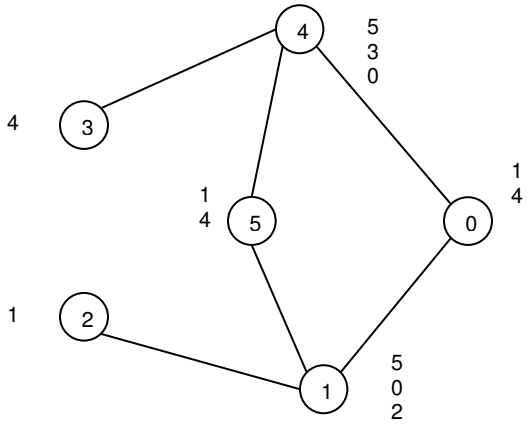
Las topologías por defecto que proporciona la aplicación son:



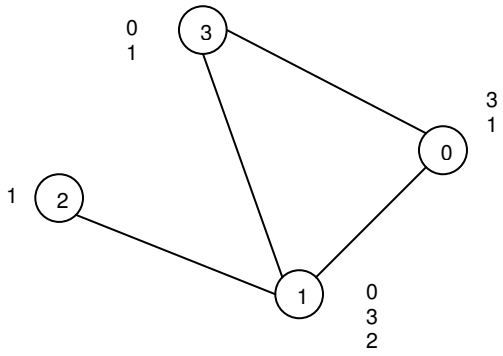
Topologías/Grafo_con_ciclo1.top



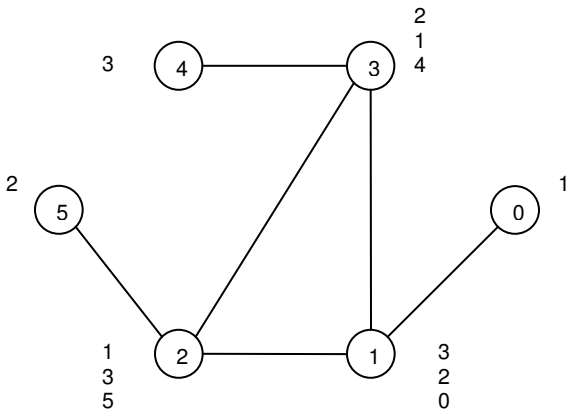
Topologías/Grafo_con_ciclo2.top



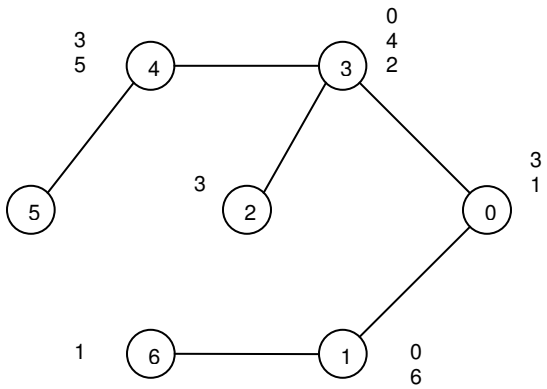
Topologías/Grafo_con_ciclo3.top



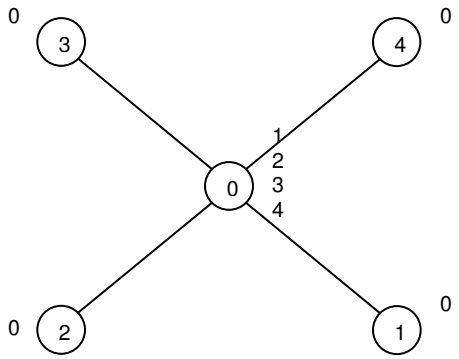
Topologías/Grafo_con_ciclo4.top



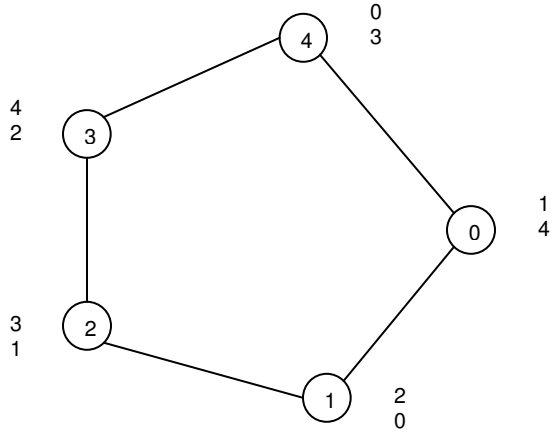
Topologías/Grafo_con_ciclo5.top



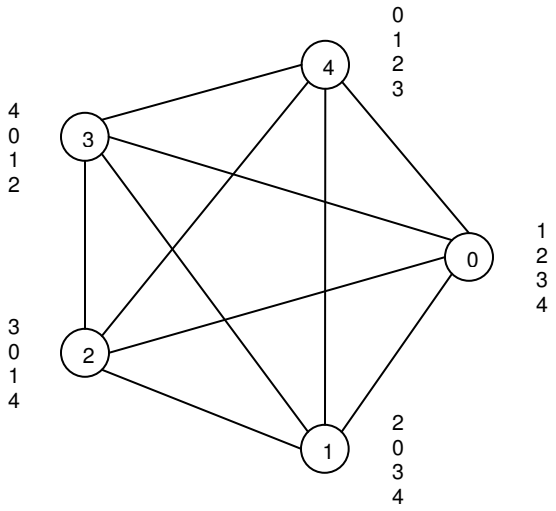
Topologías/Grafo_sin_ciclo1.top



Topologías/Estrella1.top



Topologías/Anillo1.top



Topologías/Malla_completa1.top

CONCLUSIÓN

El uso de materiales enriquecidos con imágenes, animaciones y videos produce marcados efectos en los sentidos y en la construcción de los conocimientos. Si bien el proceso cognoscitivo entabla nuevas relaciones entre los conceptos ya conocidos, el hecho que los conceptos estén traducidos para estímulos sensitivos significa facilitar el mecanismo de detectar los objetos y esquemas mentales asociados a lo que se va a aprender.

En este Trabajo Especial de Grado se ha creado un soporte didáctico que trata ser en lo posible general dentro de su contexto, para que no sólo sirva de materia prima para producir meras construcciones mentales de los conceptos sino que pueda ser modelo para el desarrollo de otros instrumentos didácticos. Así, quien desee experimentar con sistemas de animación puede encontrar en este trabajo desarrollos y ejemplos de provecho.

En cuanto al desarrollo del sistema, se ha tratado de enfocar la solución a un software lo más amplio y simple posible de modo que sea una implantación aislada del método de detección de terminación, de manera que el algoritmo base es en efecto un algoritmo general de trabajo para procesadores distribuidos que realizan múltiples tareas, presentando así los esquemas de desarrollo de los métodos de detección de terminación.

Al explicar las técnicas de detección de terminación se han mantenido los algoritmos dentro del margen de sus características esenciales, tratando de emular sus propiedades básicas en todo momento, desde que inician las animaciones hasta que se detecta la terminación general. Los autores de los algoritmos proponen alternativas y optimizaciones sobre los algoritmos los cuales han

tratado de ser introducidos aquí, sin embargo, tratamos de mantenerlos lo más sencillos posibles para que puedan darse a discusión y a la vez que no incurran en artificios técnicos provenientes de técnicas evidentemente distintas, por lo que se ha hecho una clasificación lo más segregada posible. La literatura referencial nos muestra que los algoritmos pueden mejorarse conociendo algunos hechos lógicos y matemáticos e inclusive si se mezclan entre ellos.

Después de la programación de los métodos y de aplicar una evaluación al software por distintos usuarios, hemos visto que en realidad los métodos son más complejos de lo que aparentan ser, y en realidad la presentación del modelo es una gran abstracción de todo lo requerido para hacer funcionar el algoritmo, el usuario final no notará esto, pero lo importante es que note la gran diversidad de formas en que puede resolverse un problema cuya existencia tampoco resulta tan evidente.

REFERENCIAS

[CV90] S. Chandrasekaran y S. Venkatesan. A Message-Optimal Algorithm for Distributed Termination Detection. *Journal of Parallel and Distributed Computation*, volumen 8, 1990.

[DDE04] Ronald F. DeMara, Kenneth Drake y Abdel Ejnioui. Performance Evaluation of Hierarchy Annotation and Credit Distribution Quiescence Mechanisms. *Distributed Computing*, Noviembre 2004.

[DFvG83] Edsger W. Dijkstra, W. H. J. Feijen, A. J. M. van Gasteren. Derivation of a Termination Detection Algorithm for Distributed Computations. *Information Processing Letters*, volumen 16, número 5, Junio 1983.

[DLT92] X. Dong, T.-H. Lai y Y.-C. Tseng. A more efficient message-optimal algorithm for distributed termination detection. *Proc. Sixth International Parallel Processing Symposium*, 1992.

[DTE07] R. DeMara, Y. Tseng, A. Ejnioui. Tiered Algorithm for Distributed Process Quiescence and Termination Detection. *IEEE Transactions on Parallel and Distributed Systems*, volumen 18, número 11, pág. 1529-1538, Noviembre 2007.

[DS80] E. W. Dijkstra y C. S. Scholten. Termination Detection for Diffusing Computations. *Information Processing Letters*, volumen 11, número 1, Agosto 1980.

[Mat87] Friedemann Mattern. Algorithms for Distributed Termination Detection. *Distributed Computing 2*, pág. 161 – 175, 1987.

[Mat89] Friedemann Mattern. Global Quiescence Detection based on Credit Distribution and Discovery. *Information Processing Letters*, volumen 30, número 4, Febrero 1989.

[MT90] Friedemann Mattern. Distributed Termination Detection with Sticky State Indicators. *Technical Report 200/90*, Department of Computer Science, University of Kaiserslautern, Germany, Julio 1990.

[MT91] Friedemann Mattern y Gerard Tel. The Derivation of Distributed Termination Detection Algorithms from Garbage Collection Schemes. *Proceedings on Parallel Architectures and Languages*, pág. 137-149, 1991.

[SF86] N. Shavit y N. Francez. A New Approach to Detection of Locally Indicative Stability. *Proceedings ICALP 1986, Lecture Notes in Computer Science 226*, 1986.

[Top84] R.W. Topor. Termination Detection for Distributed Computations. *Information Processing Letters*, volumen 18, número 1, pág. 33 – 36, 1984.

ANEXOS

I.- PRUEBAS COMPARATIVAS DE LOS MÉTODOS

PRUEBA I

Corrida de los distintos métodos sobre una topología de grafo con ciclo y una topología de árbol, escogencia de 2 nodos distintos y 20 procesos iniciales en memoria en 52 ejecuciones. Se contabilizaron los números de mensajes de control necesarios para detectar las terminaciones.

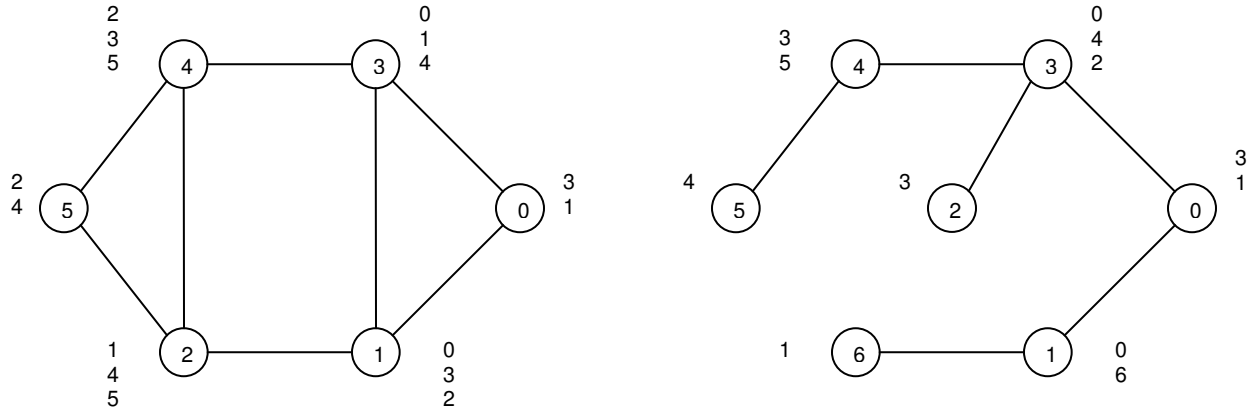


Tabla 1: Grafo con ciclos

Método	Coord.	Proc.	Msjs.		Método	Coord.	Proc.	Msjs.
Espera de Resp.	0	20	43		Espera de Resp.	2	20	35
Espera de Resp.	0	20	37		Espera de Resp.	2	20	38
Espera de Resp.	0	20	43		Espera de Resp.	2	20	40
Espera de Resp.	0	20	49		Espera de Resp.	2	20	42
Espera de Resp.	0	20	42		Espera de Resp.	2	20	41
Espera de Resp.	0	20	42		Espera de Resp.	2	20	34
Espera de Resp.	0	20	36		Espera de Resp.	2	20	39
Espera de Resp.	0	20	38		Espera de Resp.	2	20	41

Espera de Resp.	0	20	42		Espera de Resp.	2	20	37
Espera de Resp.	0	20	41		Espera de Resp.	2	20	37
Espera de Resp.	0	20	41		Espera de Resp.	2	20	39
Espera de Resp.	0	20	42		Espera de Resp.	2	20	44
Espera de Resp.	0	20	43		Espera de Resp.	2	20	36

Tabla 2: Árbol

Método	**Coord.**	**Proc.**	**Msjs.**		**Método**	**Coord.**	**Proc.**	**Msjs.**
Espera de Resp.	0	20	26		Espera de Resp.	5	20	29
Espera de Resp.	0	20	26		Espera de Resp.	5	20	22
Espera de Resp.	0	20	27		Espera de Resp.	5	20	26
Espera de Resp.	0	20	27		Espera de Resp.	5	20	29
Espera de Resp.	0	20	30		Espera de Resp.	5	20	28
Espera de Resp.	0	20	27		Espera de Resp.	5	20	29
Espera de Resp.	0	20	27		Espera de Resp.	5	20	27
Espera de Resp.	0	20	28		Espera de Resp.	5	20	28
Espera de Resp.	0	20	26		Espera de Resp.	5	20	27
Espera de Resp.	0	20	25		Espera de Resp.	5	20	28
Espera de Resp.	0	20	25		Espera de Resp.	5	20	28
Espera de Resp.	0	20	24		Espera de Resp.	5	20	29
Espera de Resp.	0	20	26		Espera de Resp.	5	20	24

Tabla 3: Grafo con ciclos

Método	**Coord.**	**Proc.**	**Msjs.**		**Método**	**Coord.**	**Proc.**	**Msjs.**
Propag. Consulta	0	20	34		Propag. Consulta	5	20	26
Propag. Consulta	0	20	34		Propag. Consulta	5	20	26
Propag. Consulta	0	20	34		Propag. Consulta	5	20	26
Propag. Consulta	0	20	34		Propag. Consulta	5	20	26
Propag. Consulta	0	20	34		Propag. Consulta	5	20	26
Propag. Consulta	0	20	34		Propag. Consulta	5	20	26
Propag. Consulta	0	20	34		Propag. Consulta	5	20	26
Propag. Consulta	0	20	34		Propag. Consulta	5	20	26

Propag. Consulta	0	20	34		Propag. Consulta	5	20	26
Propag. Consulta	0	20	34		Propag. Consulta	5	20	26
Propag. Consulta	0	20	34		Propag. Consulta	5	20	26
Propag. Consulta	0	20	34		Propag. Consulta	5	20	26
Propag. Consulta	0	20	34		Propag. Consulta	5	20	26

Tabla 4: Árbol

Método	**Coord.**	**Proc.**	**Msjs.**		**Método**	**Coord.**	**Proc.**	**Msjs.**
Propag. Consulta	0	20	36		Propag. Consulta	5	20	35
Propag. Consulta	0	20	36		Propag. Consulta	5	20	35
Propag. Consulta	0	20	36		Propag. Consulta	5	20	35
Propag. Consulta	0	20	36		Propag. Consulta	5	20	35
Propag. Consulta	0	20	36		Propag. Consulta	5	20	35
Propag. Consulta	0	20	36		Propag. Consulta	5	20	35
Propag. Consulta	0	20	36		Propag. Consulta	5	20	35
Propag. Consulta	0	20	36		Propag. Consulta	5	20	35
Propag. Consulta	0	20	36		Propag. Consulta	5	20	35
Propag. Consulta	0	20	36		Propag. Consulta	5	20	35
Propag. Consulta	0	20	36		Propag. Consulta	5	20	35
Propag. Consulta	0	20	36		Propag. Consulta	5	20	35
Propag. Consulta	0	20	36		Propag. Consulta	5	20	35

Tabla 5: Grafo con ciclos

Método	Coord.	Proc.	Msjs.		Método	Coord.	Proc.	Msjs.
Créditos	0	20	23		Créditos	2	20	27
Créditos	0	20	30		Créditos	2	20	21
Créditos	0	20	28		Créditos	2	20	25
Créditos	0	20	23		Créditos	2	20	20
Créditos	0	20	29		Créditos	2	20	24
Créditos	0	20	28		Créditos	2	20	24
Créditos	0	20	26		Créditos	2	20	21
Créditos	0	20	28		Créditos	2	20	22
Créditos	0	20	24		Créditos	2	20	21

Créditos	0	20	27		Créditos	2	20	21
Créditos	0	20	23		Créditos	2	20	26
Créditos	0	20	27		Créditos	2	20	22
Créditos	0	20	27		Créditos	2	20	21

Tabla 6: Árbol

Método	Coord.	Proc.	Msjs.		Método	Coord.	Proc.	Msjs.
Créditos	0	20	17		Créditos	5	20	31
Créditos	0	20	22		Créditos	5	20	29
Créditos	0	20	18		Créditos	5	20	29
Créditos	0	20	21		Créditos	5	20	28
Créditos	0	20	18		Créditos	5	20	22
Créditos	0	20	23		Créditos	5	20	29
Créditos	0	20	18		Créditos	5	20	32
Créditos	0	20	16		Créditos	5	20	31
Créditos	0	20	24		Créditos	5	20	26
Créditos	0	20	20		Créditos	5	20	26
Créditos	0	20	19		Créditos	5	20	29
Créditos	0	20	20		Créditos	5	20	27
Créditos	0	20	21		Créditos	5	20	33

Tabla 7: Grafo con ciclos

Método	Coord.	Proc.	Msjs.		Método	Coord.	Proc.	Msjs.
Produc / Consum	0	20	28		Produc / Consum	2	20	22
Produc / Consum	0	20	30		Produc / Consum	2	20	20
Produc / Consum	0	20	28		Produc / Consum	2	20	18
Produc / Consum	0	20	25		Produc / Consum	2	20	23
Produc / Consum	0	20	29		Produc / Consum	2	20	20
Produc / Consum	0	20	23		Produc / Consum	2	20	21
Produc / Consum	0	20	31		Produc / Consum	2	20	19
Produc / Consum	0	20	31		Produc / Consum	2	20	22
Produc / Consum	0	20	31		Produc / Consum	2	20	22

Produc / Consum	0	20	31		Produc / Consum	2	20	18
Produc / Consum	0	20	23		Produc / Consum	2	20	25
Produc / Consum	0	20	28		Produc / Consum	2	20	21
Produc / Consum	0	20	29		Produc / Consum	2	20	21

Tabla 8: Árbol

Método	Coord.	Proc.	Msjs.		Método	Coord.	Proc.	Msjs.
Produc / Consum	0	20	28		Produc / Consum	5	20	24
Produc / Consum	0	20	22		Produc / Consum	5	20	24
Produc / Consum	0	20	22		Produc / Consum	5	20	26
Produc / Consum	0	20	18		Produc / Consum	5	20	28
Produc / Consum	0	20	23		Produc / Consum	5	20	23
Produc / Consum	0	20	19		Produc / Consum	5	20	27
Produc / Consum	0	20	20		Produc / Consum	5	20	25
Produc / Consum	0	20	20		Produc / Consum	5	20	25
Produc / Consum	0	20	22		Produc / Consum	5	20	26
Produc / Consum	0	20	18		Produc / Consum	5	20	22
Produc / Consum	0	20	20		Produc / Consum	5	20	27
Produc / Consum	0	20	19		Produc / Consum	5	20	25
Produc / Consum	0	20	20		Produc / Consum	5	20	26

PRUEBA II

Ejecución del método de Propagación de Consulta con reinicio del ciclo en cualquier nodo sobre dos topologías con ciclos. Selección de distintos coordinadores iniciales y 20 procesos cargados en memoria al iniciar. Se contabilizó la cantidad de mensajes de control transmitidos en el total de ciclos antes de detectar la terminación.

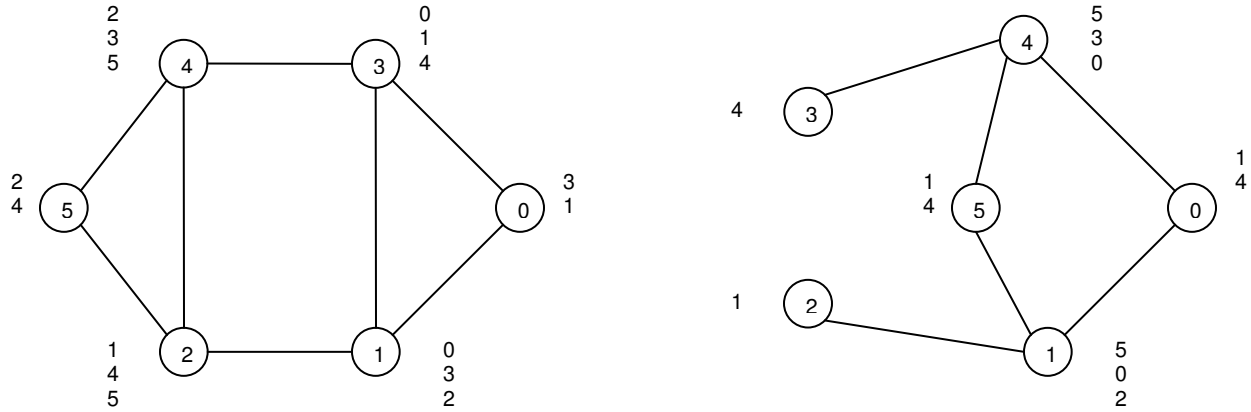


Tabla 9

Topología	Coord.	Proc.	Msjs.
Grafo con ciclo 1	0	20	34
Grafo con ciclo 1	1	20	41
Grafo con ciclo 1	2	20	37
Grafo con ciclo 1	3	20	34
Grafo con ciclo 1	4	20	29
Grafo con ciclo 1	5	20	26
Grafo con ciclo 2	0	20	28
Grafo con ciclo 2	1	20	29
Grafo con ciclo 2	2	20	30
Grafo con ciclo 2	3	20	30
Grafo con ciclo 2	4	20	31
Grafo con ciclo 2	5	20	30

PRUEBA III

Ejecución de los métodos de detección sobre una topología en forma de anillo. Coordinador inicial fijo y 20 procesos locales en memoria al iniciar. Se contabilizó el total de mensajes de control utilizados antes de detectar la terminación.

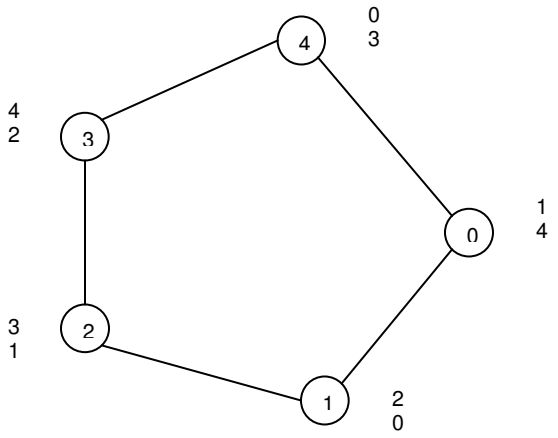


Tabla 10

Método	Coord.	Proc.	Msjs.		Método	Coord.	Proc.	Msjs.
Créditos	0	20	21		Esp. de respuesta	0	20	28
Créditos	0	20	14		Esp. de respuesta	0	20	30
Créditos	0	20	26		Esp. de respuesta	0	20	31
Créditos	0	20	21		Esp. de respuesta	0	20	33
Créditos	0	20	22		Esp. de respuesta	0	20	34
Créditos	0	20	17		Esp. de respuesta	0	20	35
Produc / Consum	0	20	22		Prop. de Consulta	0	20	14
Produc / Consum	0	20	26		Prop. de Consulta	0	20	14
Produc / Consum	0	20	23		Prop. de Consulta	0	20	14
Produc / Consum	0	20	23		Prop. de Consulta	0	20	14
Produc / Consum	0	20	26		Prop. de Consulta	0	20	14
Produc / Consum	0	20	23		Prop. de Consulta	0	20	14

II.- ÉXAMEN DE APLICACIÓN CON USUARIOS

Usuario: Rommel Quiñones

Perfil: Ingeniero en Informática

1) ¿Utilizó con facilidad el software?

Sí.

2) ¿Cuáles algoritmos visualizó usted?

Todos.

3) ¿Conocía usted previamente alguno de los algoritmos animados?

No.

4) ¿Comprendió las animaciones? ¿Comprendió los algoritmos simulados? ¿Fueron las animaciones suficientemente claras mostrando la idea de los algoritmos?

Sí.

5) ¿Le parecieron triviales los algoritmos animados?

Algunos.

6) ¿Cree que es necesario conocer previamente los algoritmos para comprender las animaciones?

Sí.

7) ¿Terminaron las animaciones iniciadas? ¿Finalizaron todos los procesos al decretarse la finalización general del sistema?

Sí.

8) ¿Tardaron mucho las animaciones para finalizar?

Más o menos.

9) ¿Configuró usted la animación? ¿Probó alterar alguna de las variables configurables de la animación?

Sí, algunos.

10) ¿Probó algún algoritmo más de una vez? De ser así ¿Notó diferencias entre los distintos intentos?

Sí, pero ninguna diferencia notoria.

11) ¿Quedó usted satisfecho con la estocástica del sistema?

Sí.

12) ¿Recomienda usted agregarle algo a la aplicación? ¿Cambiaría o quitaría algo a las animaciones?

Me parece que está bien.

13) ¿Cree que los elementos de animación distraen la atención? ¿Cree que la interfaz de animación desvirtúa el propósito del software?

Entretiene mas no distrae.

14) ¿En algún momento le pareció tediosa la animación?

No.

Usuario: Juan Leung

Perfil: Licenciado en Informática

1) ¿Utilizó con facilidad el software?

Sí.

2) ¿Cuáles algoritmos visualizó usted?

Todos.

3) ¿Conocía usted previamente alguno de los algoritmos animados?

No.

4) ¿Comprendió las animaciones? ¿Comprendió los algoritmos simulados? ¿Fueron las animaciones suficientemente claras mostrando la idea de los algoritmos?

Sí, sí, sí.

5) ¿Le parecieron triviales los algoritmos animados?

No..

6) ¿Cree que es necesario conocer previamente los algoritmos para comprender las animaciones?

Sí.

7) ¿Terminaron las animaciones iniciadas? ¿Finalizaron todos los procesos al decretarse la finalización general del sistema?

Sí, sí.

8) ¿Tardaron mucho las animaciones para finalizar?

Sí.

9) ¿Configuró usted la animación? ¿Probó alterar alguna de las variables configurables de la animación?

Sí, sí.

10) ¿Probó algún algoritmo más de una vez? De ser así ¿Notó diferencias entre los distintos intentos?

Sí, sí. Más tiempo.

11) ¿Quedó usted satisfecho con la estocástica del sistema?

Sí.

12) ¿Recomienda usted agregarle algo a la aplicación? ¿Cambiaría o quitaría algo a las animaciones?

Sí, un sombrerito al coordinador.

13) ¿Cree que los elementos de animación distraen la atención? ¿Cree que la interfaz de animación desvirtúa el propósito del software?

No, no, más bien facilita la comprensión de los algoritmos.

14) ¿En algún momento le pareció tediosa la animación?

No.

Usuario: Joanna Cabezas

Perfil: Estudiante de Computación

1) ¿Utilizó con facilidad el software?

Sí.

2) ¿Cuáles algoritmos visualizó usted?

Todos.

3) ¿Conocía usted previamente alguno de los algoritmos animados?

No.

4) ¿Comprendió las animaciones? ¿Comprendió los algoritmos simulados? ¿Fueron las animaciones suficientemente claras mostrando la idea de los algoritmos?

Sí.

5) ¿Le parecieron triviales los algoritmos animados?

No.

6) ¿Cree que es necesario conocer previamente los algoritmos para comprender las animaciones?

Sí.

7) ¿Terminaron las animaciones iniciadas? ¿Finalizaron todos los procesos al decretarse la finalización general del sistema?

Sí.

8) ¿Tardaron mucho las animaciones para finalizar?

Sólo un poco.

9) ¿Configuró usted la animación? ¿Probó alterar alguna de las variables configurables de la animación?

Sí.

10) ¿Probó algún algoritmo más de una vez? De ser así ¿Notó diferencias entre los distintos intentos?

Sí. No hubo diferencias notables.

11) ¿Quedó usted satisfecho con la estocástica del sistema?

Sí.

12) ¿Recomienda usted agregarle algo a la aplicación? ¿Cambiaría o quitaría algo a las animaciones?

Cambiaría la música por una más calmada.

13) ¿Cree que los elementos de animación distraen la atención? ¿Cree que la interfaz de animación desvirtúa el propósito del software?

La música distrae un poco.

14) ¿En algún momento le pareció tediosa la animación?

No.

Usuario: Roberto Meléndez

Perfil: Estudiante de Computación

1) ¿Utilizó con facilidad el software?

Sí.

2) ¿Cuáles algoritmos visualizó usted?

Todos.

3) ¿Conocía usted previamente alguno de los algoritmos animados?

Sí, algunos.

4) ¿Comprendió las animaciones? ¿Comprendió los algoritmos simulados? ¿Fueron las animaciones suficientemente claras mostrando la idea de los algoritmos?

Sí.

5) ¿Le parecieron triviales los algoritmos animados?

No.

6) ¿Cree que es necesario conocer previamente los algoritmos para comprender las animaciones?

Un poco.

7) ¿Terminaron las animaciones iniciadas? ¿Finalizaron todos los procesos al decretarse la finalización general del sistema?

Sí.

8) ¿Tardaron mucho las animaciones para finalizar?

No.

9) ¿Configuró usted la animación? ¿Probó alterar alguna de las variables configurables de la animación?

Sí.

10) ¿Probó algún algoritmo más de una vez? De ser así ¿Notó diferencias entre los distintos intentos?

Sí.

11) ¿Quedó usted satisfecho con la estocástica del sistema?

Sí.

12) ¿Recomienda usted agregarle algo a la aplicación? ¿Cambiaría o quitaría algo a las animaciones?

Hace falta resaltar un poco más el coordinador.

13) ¿Cree que los elementos de animación distraen la atención? ¿Cree que la interfaz de animación desvirtúa el propósito del software?

No.

14) ¿En algún momento le pareció tediosa la animación?

No.

Usuario: José Chang

Perfil: Licenciado en Computación

1) ¿Utilizó con facilidad el software?

Sí.

2) ¿Cuáles algoritmos visualizó usted?

Todos.

3) ¿Conocía usted previamente alguno de los algoritmos animados?

No.

4) ¿Comprendió las animaciones? ¿Comprendió los algoritmos simulados? ¿Fueron las animaciones suficientemente claras mostrando la idea de los algoritmos?

Sí.

5) ¿Le parecieron triviales los algoritmos animados?

No.

6) ¿Cree que es necesario conocer previamente los algoritmos para comprender las animaciones?

Sí.

7) ¿Terminaron las animaciones iniciadas? ¿Finalizaron todos los procesos al decretarse la finalización general del sistema?

Sí.

8) ¿Tardaron mucho las animaciones para finalizar?

No.

9) ¿Configuró usted la animación? ¿Probó alterar alguna de las variables configurables de la animación?

Sí.

10) ¿Probó algún algoritmo más de una vez? De ser así ¿Notó diferencias entre los distintos intentos?

Sí. Sí.

11) ¿Quedó usted satisfecho con la estocástica del sistema?

Sí.

12) ¿Recomienda usted agregarle algo a la aplicación? ¿Cambiaría o quitaría algo a las animaciones?

No.

13) ¿Cree que los elementos de animación distraen la atención? ¿Cree que la interfaz de animación desvirtúa el propósito del software?

No.

14) ¿En algún momento le pareció tediosa la animación?

No.

Usuario: Hernán Albornoz

Perfil: Ingeniero en Informática

1) ¿Utilizó con facilidad el software?

Sí.

2) ¿Cuáles algoritmos visualizó usted?

Todos.

3) ¿Conocía usted previamente alguno de los algoritmos animados?

Sí, el de Diskjtra y el de créditos.

4) ¿Comprendió las animaciones? ¿Comprendió los algoritmos simulados? ¿Fueron las animaciones suficientemente claras mostrando la idea de los algoritmos?

Es un poco más complejo de lo que esperaba.

5) ¿Le parecieron triviales los algoritmos animados?

El de créditos.

6) ¿Cree que es necesario conocer previamente los algoritmos para comprender las animaciones?

Definitivamente.

7) ¿Terminaron las animaciones iniciadas? ¿Finalizaron todos los procesos al decretarse la finalización general del sistema?

Sí.

8) ¿Tardaron mucho las animaciones para finalizar?

A veces sí.

9) ¿Configuró usted la animación? ¿Probó alterar alguna de las variables configurables de la animación?

Sí, todos.

10) ¿Probó algún algoritmo más de una vez? De ser así ¿Notó diferencias entre los distintos intentos?

Sí.

11) ¿Quedó usted satisfecho con la estocástica del sistema?

Sí.

12) ¿Recomienda usted agregarle algo a la aplicación? ¿Cambiaría o quitaría algo a las animaciones?

Poder quitar la música. Se necesita de un tutor para aprender.

13) ¿Cree que los elementos de animación distraen la atención? ¿Cree que la interfaz de animación desvirtúa el propósito del software?

Quizás la música a veces.

14) ¿En algún momento le pareció tediosa la animación?

Me divertí.