Universidad del Valle

Paradigmas Avanzados de Programación

Sistemas Multi Agentes

Juan Francisco Díaz Frias

Maestría en Ingeniería, Énfasis en Ingeniería de Sistemas y Computación
Escuela de Ingeniería de Sistemas y Computación,
home page: http://eisc.univalle.edu.co
Universidad del Valle - Cali, Colombia

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Programación con componentes concurrentes

Programación basada en componentes

- Componentes concurrentes.
- Sin la limitación de la sincronía.

Primero: modelar

- Una aplicacion se modela como un conjunto de actividades concurrentes que interactúan en formas bien definidas.
- Cada actividad concurrente se modela exáctamente con un componente concurrente.
- Componente concurrente $\equiv$ agente.

Programación con componentes concurrentes

Programación basada en componentes

- Componentes concurrentes.
- Sin la limitación de la sincronía.

Primero: modelar

- Una aplicacion se modela como un conjunto de actividades concurrentes que interactúan en formas bien definidas.
- Cada actividad concurrente se modela exáctamente con un componente concurrente.
- Componente concurrente $\equiv$ agente.

Sistemas Multi Agentes: MAS

Algunas características

- Los agentes pueden ser reactivos (no tienen estado interno) o tener un estado interno.
- En la programación basada en componentes, los agentes son considerados, normalmente, entidades bastante simples con poca inteligencia incorporada.
- En la comunidad de inteligencia artificial, se considera que los agentes, normalmente, realizan el mismo tipo de razonamiento.

Un modelo para la programación con componentes concurrentes

- Componentes primitivos.
- Formas de combinar componentes.

Sistemas Multi Agentes: MAS

Algunas características

- Los agentes pueden ser reactivos (no tienen estado interno) o tener un estado interno.
- En la programación basada en componentes, los agentes son considerados, normalmente, entidades bastante simples con poca inteligencia incorporada.
- En la comunidad de inteligencia artificial, se considera que los agentes, normalmente, realizan el mismo tipo de razonamiento.

Un modelo para la programación con componentes concurrentes

- Componentes primitivos.
- Formas de combinar componentes.

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Un modelo sencillo para programación basada en componentes (1)

El modelo

- Basado en objetos puerto y ejecutado con concurrencia por paso de mensajes.
- Un componente concurrente es un procedimiento con entradas y salidas.
- Cuando se invoca, el componente crea una instancia del componente, la cual es un objeto puerto.
- Una entrada es un puerto cuyo flujo es leido por el componente.
- Una salida es un puerto por el cual el componente puede enviar mensajes.

Un modelo sencillo para programación basada en componentes (2)

Procedimientos como componentes concurrentes

- Ellos son composicionales.
- Ellos pueden tener un número arbitrario de entradas y salidas.
- Cuando se componen los subcomponentes, ellos permiten la visibilidad de las entradas y las salidas que deben ser controladas (por ejemplo, algunos pueden tener visibilidad restringida al interior del componente).

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Interfaz (1)

Interfaz $\equiv$ Medio de interacción con el ambiente

La interfaz consiste del conjunto de entradas y salidas del componente, las cuales se conocen colectivamente como sus cables.

Cables

- Conecta una o más salidas a una o más entradas.
- Dos tipos básicos de cables: de un tiro y de muchos tiros.

Interfaz (1)

Interfaz $\equiv$ Medio de interacción con el ambiente

La interfaz consiste del conjunto de entradas y salidas del componente, las cuales se conocen colectivamente como sus cables.

Cables

- Conecta una o más salidas a una o más entradas.
- Dos tipos básicos de cables: de un tiro y de muchos tiros.

Interfaz (2)

Cables de un tiro

- Se implementan con variables de flujo de datos.
- Se usan para valores que no cambian o para mensajes por una sola vez (como acuses de recibo).
- Solamente se puede pasar un mensaje, y sólo se puede conectar una salida a una entrada dada.

Cables de muchos tiros

- Se implementan con puertos.
- Se utilizan para flujos de mensajes.
- Se puede enviar cualquier número de mensajes, y cualquier número de salidas pueden escribir en una entrada dada.

Interfaz (2)

Cables de un tiro

- Se implementan con variables de flujo de datos.
- Se usan para valores que no cambian o para mensajes por una sola vez (como acuses de recibo).
- Solamente se puede pasar un mensaje, y sólo se puede conectar una salida a una entrada dada.

Cables de muchos tiros

- Se implementan con puertos.
- Se utilizan para flujos de mensajes.
- Se puede enviar cualquier número de mensajes, y cualquier número de salidas pueden escribir en una entrada dada.

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Operaciones básicas programación basada en componentes

Instanciación
Creación de una instancia de un componente. Cada instancia es independiente de las otras.

Composición
Construcción de un nuevo componente a partir de otros componentes. Los componentes que deseamos componer son independientes.

Acoplamiento
Combinación de instancias de componentes, por medio de la conexión de entradas y salidas de unos y otros.

Restricción
Restricción de la visibilidad de las entradas o las salidas dentro de un componente compuesto.

Operaciones básicas programación basada en componentes

Instanciación
Creación de una instancia de un componente. Cada instancia es independiente de las otras.

Composición
Construcción de un nuevo componente a partir de otros componentes. Los componentes que deseamos componer son independientes.

Acoplamiento
Combinación de instancias de componentes, por medio de la conexión de entradas y salidas de unos y otros.

Restricción
Restricción de la visibilidad de las entradas o las salidas dentro de un componente compuesto.

Modelos y Paradigmas de Programación

Operaciones básicas programación basada en componentes

Instanciación
Creación de una instancia de un componente. Cada instancia es independiente de las otras.

Composición
Construcción de un nuevo componente a partir de otros componentes. Los componentes que deseamos componer son independientes.

Acoplamiento
Combinación de instancias de componentes, por medio de la conexión de entradas y salidas de unos y otros.

Restricción
Restricción de la visibilidad de las entradas o las salidas dentro de un componente compuesto.

Operaciones básicas programación basada en componentes

Instanciación
Creación de una instancia de un componente. Cada instancia es independiente de las otras.

Composición
Construcción de un nuevo componente a partir de otros componentes. Los componentes que deseamos componer son independientes.

Acoplamiento
Combinación de instancias de componentes, por medio de la conexión de entradas y salidas de unos y otros.

Restricción
Restricción de la visibilidad de las entradas o las salidas dentro de un componente compuesto.

Ejemplo

Recordar la definición del componente `Cerrojo` en la simulación de circuitos digitales:

```
proc {Cerrojo C DI ?DO}
   X Y Z F
in
   {CRetraso DO F}
   {And F C X}
   {CNot C Z}
   {CAnd Z DI Y}
   {COr X Y DO}
end
```

Cinco subcomponentes, acoplados y con ciertos cables restringidos a su interior.

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Metodología de diseño (1)

Las interacciones posibles dificultan la tarea de diseño. Es importante seguir una secuencia de reglas de diseño que no sean ambiguas.

Especificación informal

. . . pero precisa.

Componentes

Enumerar todas las formas diferentes de actividad concurrente en la especificación. Cada actividad se volverá un componente. Dibujar un diagrama de bloques del sistema que muestre todas las instancias de componentes.

Metodología de diseño (1)

Las interacciones posibles dificultan la tarea de diseño. Es importante seguir una secuencia de reglas de diseño que no sean ambiguas.

Especificación informal

. . . pero precisa.

Componentes

Enumerar todas las formas diferentes de actividad concurrente en la especificación. Cada actividad se volverá un componente. Dibujar un diagrama de bloques del sistema que muestre todas las instancias de componentes.

Metodología de diseño (1)

Las interacciones posibles dificultan la tarea de diseño. Es importante seguir una secuencia de reglas de diseño que no sean ambiguas.

Especificación informal

. . . pero precisa.

Componentes

Enumerar todas las formas diferentes de actividad concurrente en la especificación. Cada actividad se volverá un componente. Dibujar un diagrama de bloques del sistema que muestre todas las instancias de componentes.

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Metodología de diseño (2)

Protocolos de mensajes

Decida qué mensajes enviarán los componentes y diseñe los protocolos de mensajes entre ellos. Dibujar el diagrama de componentes con todos los protocolos de mensajes.

Diagramas de estado

Para cada entidad concurrente, escribir su diagrama de estados. Para cada estado, verificar que se reciben y se envían todos los mensajes apropiados con las condiciones y acciones correctas.

Metodología de diseño (2)

Protocolos de mensajes

Decida qué mensajes enviarán los componentes y diseñe los protocolos de mensajes entre ellos. Dibujar el diagrama de componentes con todos los protocolos de mensajes.

Diagramas de estado

Para cada entidad concurrente, escribir su diagrama de estados. Para cada estado, verificar que se reciben y se envían todos los mensajes apropiados con las condiciones y acciones correctas.

Metodología de diseño (2)

Protocolos de mensajes

Decida qué mensajes enviarán los componentes y diseñe los protocolos de mensajes entre ellos. Dibujar el diagrama de componentes con todos los protocolos de mensajes.

Diagramas de estado

Para cada entidad concurrente, escribir su diagrama de estados. Para cada estado, verificar que se reciben y se envían todos los mensajes apropiados con las condiciones y acciones correctas.

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Metodología de diseño (3)

Implementar y planificar

Codificar el sistema en el lenguaje de programación favorito. Decidir cuál algoritmo de planificación se utilizará para implementar la concurrencia entre los componentes.

Probar e iterar

Probar el sistema y reiterar hasta que se satisfaga la especificación inicial.

Metodología de diseño (3)

Implementar y planificar

Codificar el sistema en el lenguaje de programación favorito. Decidir cuál algoritmo de planificación se utilizará para implementar la concurrencia entre los componentes.

Probar e iterar

Probar el sistema y reiterar hasta que se satisfaga la especificación inicial.

Metodología de diseño (3)

Implementar y planificar

Codificar el sistema en el lenguaje de programación favorito. Decidir cuál algoritmo de planificación se utilizará para implementar la concurrencia entre los componentes.

Probar e iterar

Probar el sistema y reiterar hasta que se satisfaga la especificación inicial.

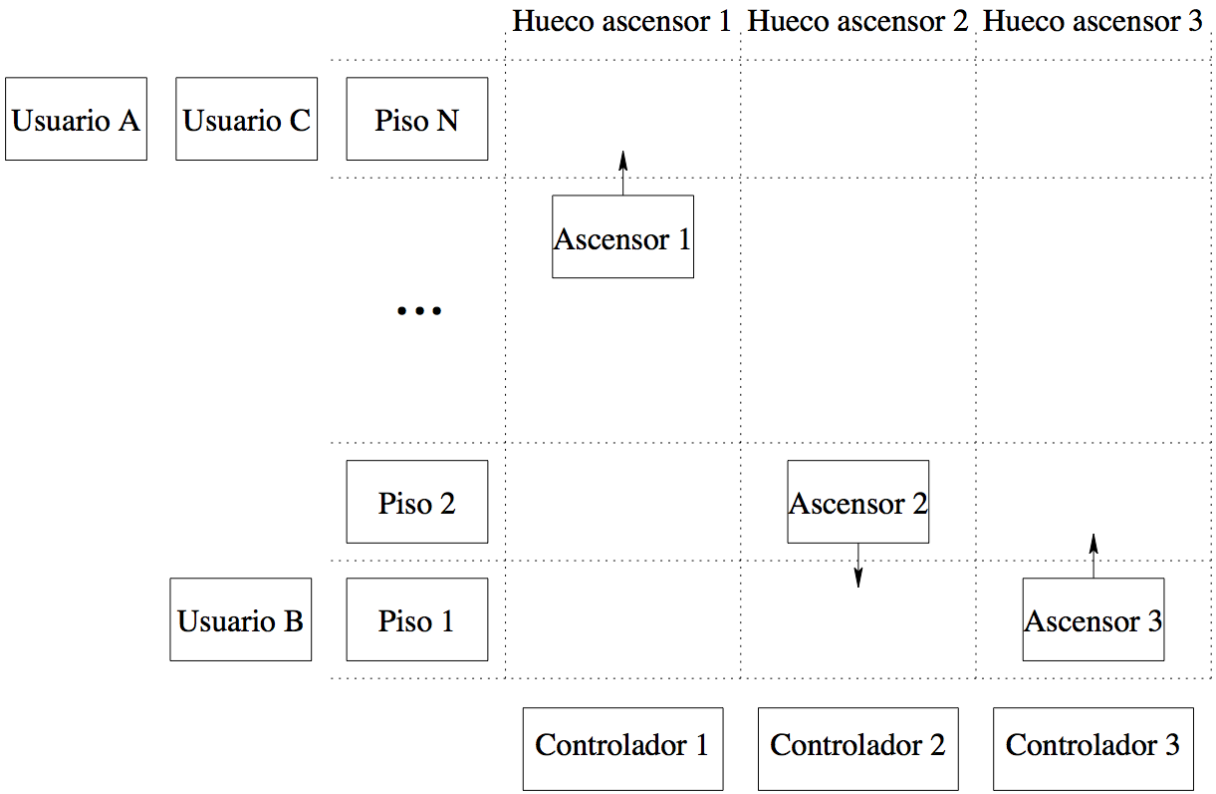
Modelos y Paradigmas de Programación

Universidad del Valle

Plan

El problema del control de ascensores

Modelaremos la operación de un sistema hipotético de control de ascensores de una edificación, con un número fijo de ascensores, un número fijo de pisos entre los cuales viajan los ascensores, y los usuarios.



Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Especificación del problema (1)

- Hay un conjunto de pisos y un conjunto de ascensores.
- Cada piso cuenta con un botón, que los usuarios pueden presionar, para llamar el ascensor.
- El botón de llamado no especifica ninguna dirección hacia arriba o hacia abajo.
- El piso escoge aleatoriamente el ascensor que le va a prestar el servicio.
- Cada ascensor tiene una serie de botones, piso(I), numerados para todos los pisos I, para solicitarle que se detenga en un piso dado.
- Cada ascensor tiene un plan, el cual consiste en la lista de los pisos que debe visitar, en el orden en que debe hacerlo.

Especificación del problema (2)

- El algoritmo de planificación que usaremos se denomina FCFS: un piso nuevo siempre se coloca al final del plan.
- Tanto el botón para llamar el ascensor desde un piso, como los botones piso(I) planifican con FCFS.
- Cuando un ascensor llega a un piso planificado, las puertas se abren y permanecen abiertas por un tiempo fijo antes de cerrarse de nuevo.
- El ascensor toma un tiempo fijo para moverse de un piso al siguiente.

Modelos y Paradigmas de Programación

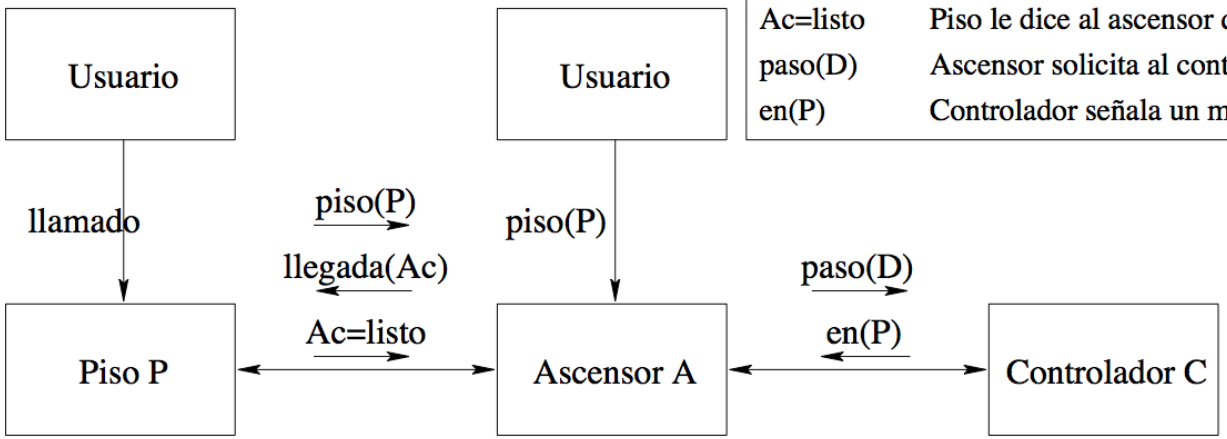
Universidad del Valle

Plan

Componentes y Protocolos de mensajes (1)

El sistema de control de ascensores se ha diseñado como un conjunto de componentes concurrentes que interactúan entre ellos.

llamado	Usuario presiona el botón para llamar el ascensor
piso(P)	Piso P solicita un ascensor en el piso Usuario presiona el botón para ir al piso P
llegada(Ac)	Ascensor señala su llegada al piso
Ac=listo	Piso le dice al ascensor que ya puede irse
paso(D)	Ascensor solicita al controlador moverse un piso
en(P)	Controlador señala un movimiento exitoso



Componentes y Protocolos de mensajes (2)

- Cada rectángulo representa una instancia de un componente concurrente.
- Cuatro tipos de componentes: pisos, ascensores, controladores, y temporizadores.
- Todas las instancias de componentes son objetos puerto.
- Los controladores se usan para manejar el movimiento de los ascensores.
- Los temporizadores manejan el aspecto de tiempo real del sistema.

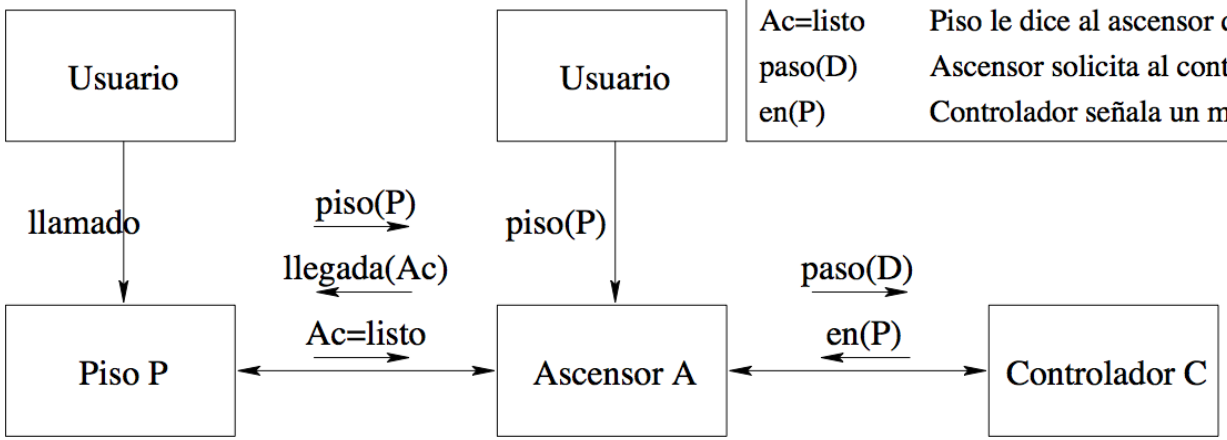
Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Notación para diagramas de estado

llamado	Usuario presiona el botón para llamar el ascensor
piso(P)	Piso P solicita un ascensor en el piso Usuario presiona el botón para ir al piso P
llegada(Ac)	Ascensor señala su llegada al piso
Ac=listo	Piso le dice al ascensor que ya puede irse
paso(D)	Ascensor solicita al controlador moverse un piso
en(P)	Controlador señala un movimiento exitoso



- Autómata de estados finitos: estados y transiciones.
- En cada instante, el componente está en un estado en particular.
- El componente se encuentra en un estado inicial, y evoluciona realizando transiciones.
- Transición: Operación atómica.

Envío de mensajes y retrasos

Envío de mensajes

De dos maneras: a un puerto o ligando una variable de flujo de datos.

Retrasos

Protocolo temporizador:

- El invocador `Pid` envía el mensaje `inictempdr(N Pid)` a un agente temporizador para solicitar un retraso de `N` ms, y continúa su trabajo.
- Cuando el tiempo ha pasado, el agente temporizador envía el mensaje de vuelta, `fintempdr`, al invocador.

Envío de mensajes y retrasos

Envío de mensajes

De dos maneras: a un puerto o ligando una variable de flujo de datos.

Retrasos

Protocolo temporizador:

- El invocador `Pid` envía el mensaje `inictempdr(N Pid)` a un agente temporizador para solicitar un retraso de `N` ms, y continúa su trabajo.
- Cuando el tiempo ha pasado, el agente temporizador envía el mensaje de vuelta, `fintempdr`, al invocador.

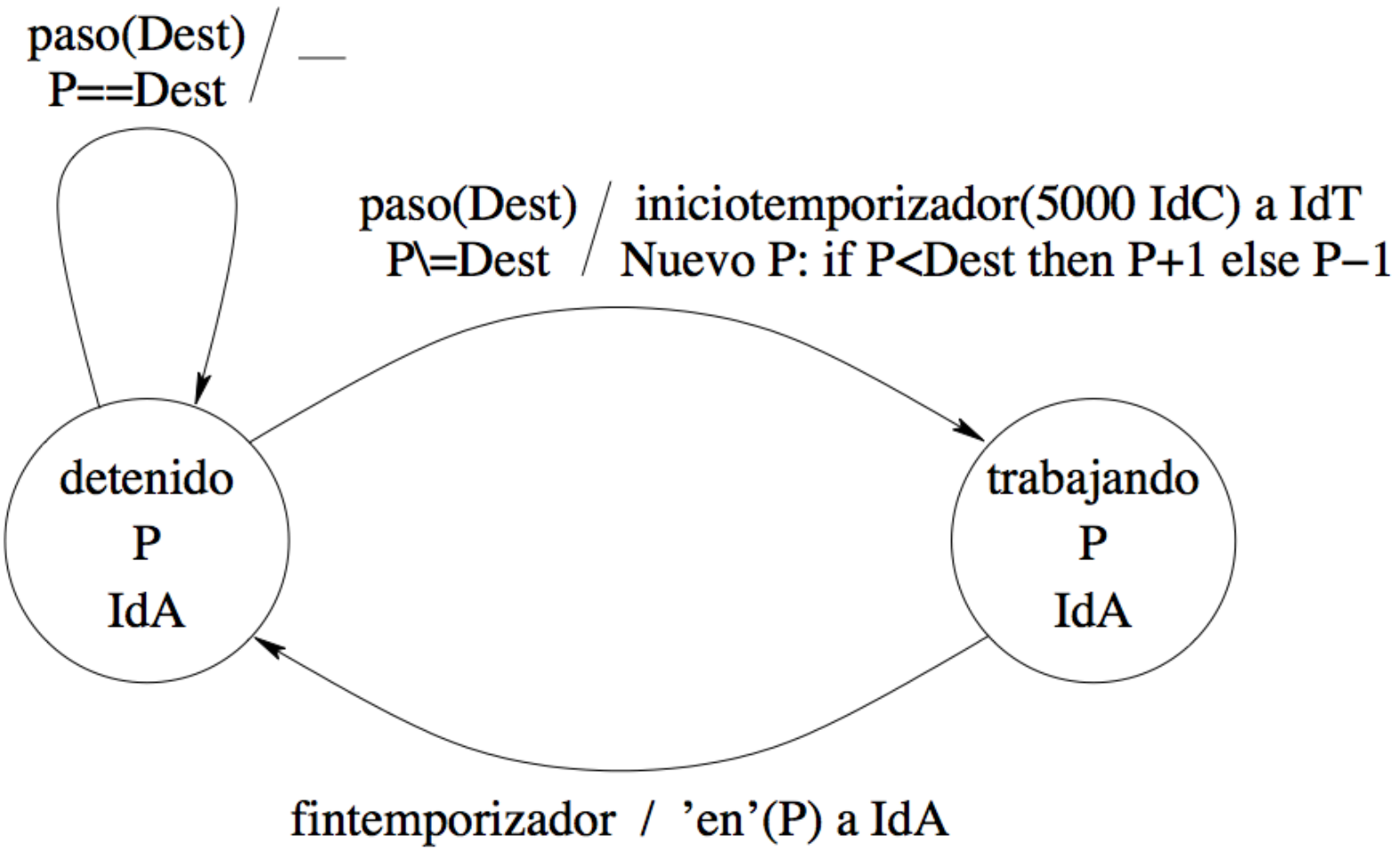
DE e Implementación de cada componente

Implementación del controlador, el piso, y el ascensor.

- `{Piso Num Inic Ascensores}` devuelve una instancia del componente piso `IdPiso`, con número `Num`, estado inicial `Inic`, y ascensores `Ascensores`.
- `{Ascensor Num Inic IdC Pisos}` devuelve una instancia del componente ascensor `IdA`, con número `Num`, estado inicial `Inic`, controlador `IdC`, y pisos `Pisos`.
- `{Controlador Inic}` devuelve una instancia del componente controlador `IdC`.

Universidad del Valle

El Controlador: DE



El Controlador: Código

```
fun {Temporizador}
   {NuevoObjetoPuerto2
    proc {$ Msj}
       case Msj of inictempdr(T IdC) then
          thread {Delay T} {Send IdC fintempdr} end
       end
    end}
end %$

fun {Controlador Inic}
   IdT={Temporizador}
   IdC={NuevoObjetoPuerto Inic ControlDE}
in IdC end
```

El Controlador: Función de transición de estados (1)

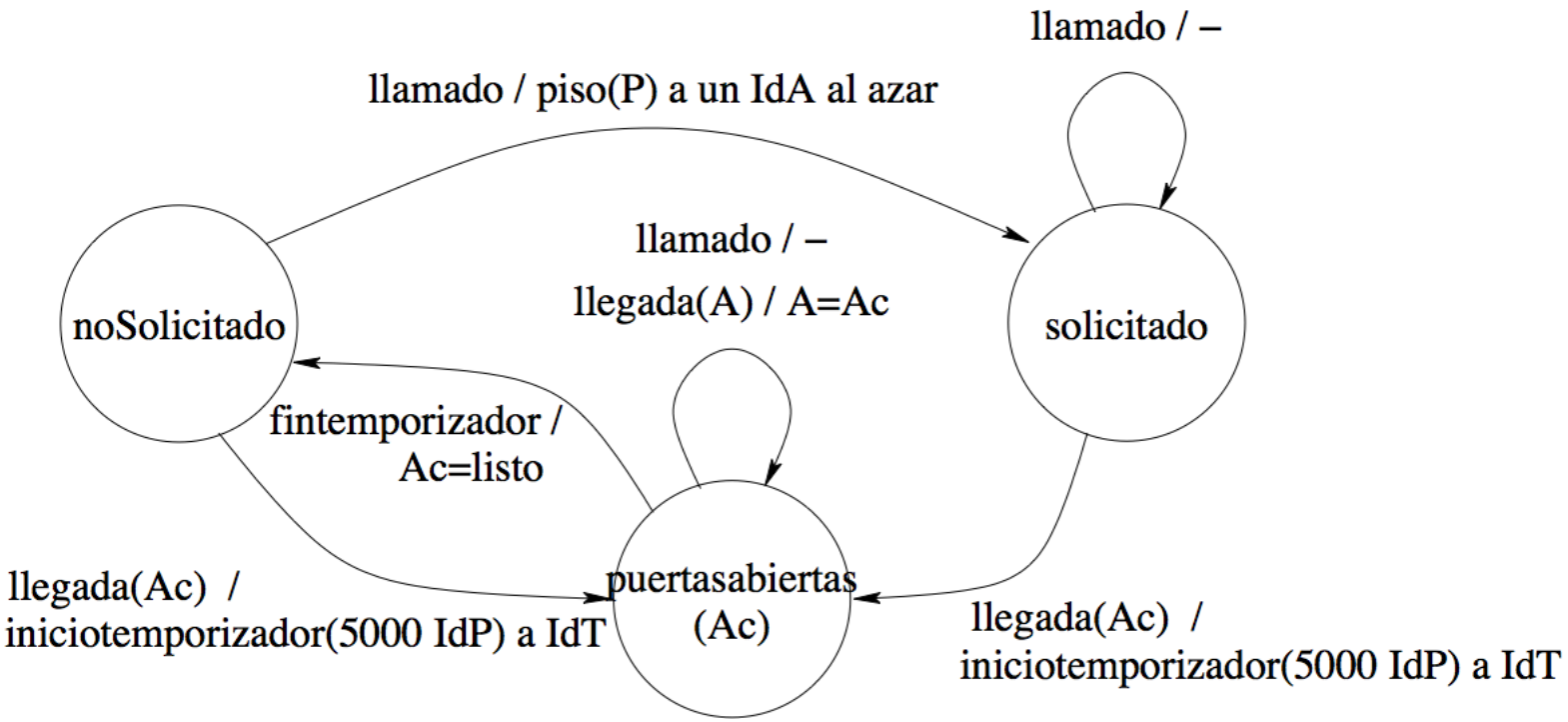
```
fun {ControlDE estado(Motor P IdA) Msj}
   case Motor
   of trabajando then
      case Msj
      of fintempdr then
         {Send IdA 'en'(P)}
         estado(detenido P IdA)
      end
   [] ...
end
```

El Controlador: Función de transición de estados (2)

```
fun {ControlDE estado(Motor P IdA) Msj}
   case Motor
   of ...
   [] detenido then
      case Msj
      of paso(Dest) then
         if P==Dest then
            estado(detenido P IdA)
         elseif P<Dest then
            {Send IdT inictempdr(5000 IdC)}
            estado(trabajando P+1 IdA)
         else % P>Dest
            {Send IdT inictempdr(5000 IdC)}
            estado(trabajando P-1 IdA)
         end
      end
   end
end
```

Modelos y Paradigmas de Programación

El Piso: DE



El Piso: Implementación

```
fun {Piso Num Inic Ascensores}
   IdT={Temporizador}
   IdPiso={NuevoObjetoPuerto Inic PisoDE}
end
```

El Piso: Función de transición de estados (1)

```
fun {PisoDE estado(EstP) Msj}
    case EstP
       of nosolicitado then AsAzar in
       case Msj
       of llegada(Ac) then
          {Browse 'Ascensor en el piso '#Num#': apertura
puertas'}
          {Send IdT inictempdr(5000 IdPiso)}
          estado(puertasabiertas(Ac))
       [] ...
```

El Piso: Función de transición de estados (2)

```
fun {PisoDE estado(EstP) Msj}
   case EstP
      of nosolicitado then AsAzar in
      case Msj
      of ...
      [] llamado then
         {Browse 'Piso '#Num#' solicita un ascensor!'}
         AsAzar=Ascensores.(1+{OS.rand} mod {Width Ascensores})
         {Send AsAzar piso(Num)}
         estado(solicitado)
      end
   ...
```

El Piso: Función de transición de estados (3)

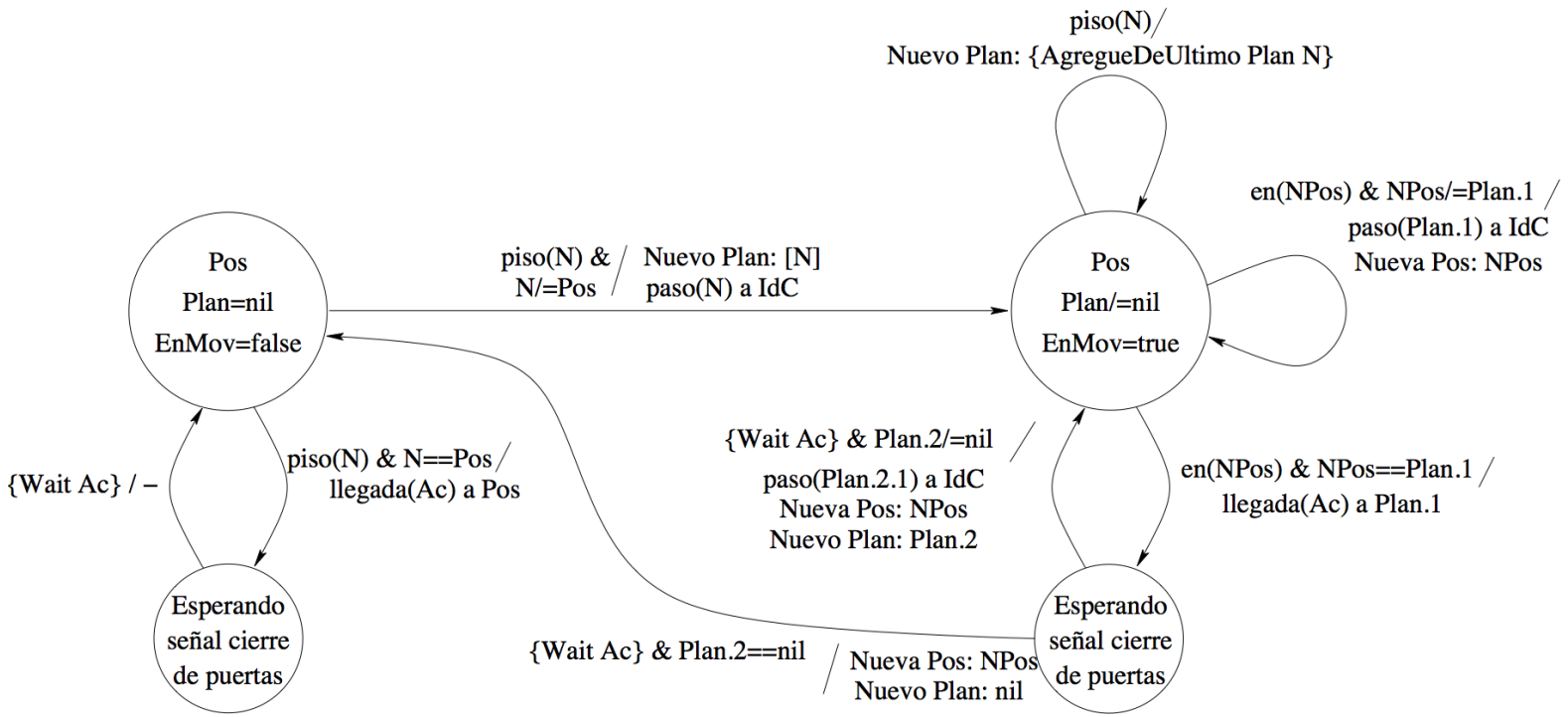
```
fun {PisoDE estado(EstP) Msj}
    case EstP
       of ...
    [] solicitado then
       case Msj
       of llegada(Ac) then
          {Browse 'Ascensor en el piso '#Num#': apertura
puertas'}
          {Send IdT inictempdr(5000 IdPiso)}
          estado(puertasabiertas(Ac))
          [] llamado then
          estado(solicitado)
       end
```

El Piso: Función de transición de estados (4)

```
fun {PisoDE estado(EstP) Msj}
   case EstP
      of ...
   [] puertasabiertas(Ac) then
      case Msj
      of fintempdr then
         {Browse 'Ascensor en el piso '#Num#': cierre puertas'}
         Ac=listo
         estado(nosolicitado)
      [] llegada(A) then
         A=Ac
         estado(puertasabiertas(Ac))
      [] llamado then
         estado(puertasabiertas(Ac))
      end
   end
end
```

Modelos y Paradigmas de Programación

El Ascensor: DE



El Ascensor: Implementación

```
fun {AgregueDeUltimo L N}
   if L\=nil andthen {List.last L}==N then L
   else {Append L [N]} end
end

fun {Ascensor Num Inic IdC Pisos}
   {NuevoObjetoPuerto Inic AscensorDE}
```

El Ascensor: Función de transición de estados (1)

```
fun {AscensorDE estado(Pos Plan EnMov) Msj}
      case Msj
      of piso(N) then
         {Browse 'Ascensor '#Num#' solicitado en el piso '#N}
         if N==Pos andthen {Not EnMov} then
            {Wait {Send Pisos.Pos llegada($)}}
            estado(Pos Plan false)
         else Plan2 in
            Plan2={AgregueDeUltimo Plan N}
            if {Not EnMov} then
               {Send IdC paso(N)} end
            estado(Pos Plan2 true)
         end
       ...
```

El Ascensor: Función de transición de estados (2)

```
fun {AscensorDE estado(Pos Plan EnMov) Msj}
      case Msj
      of piso(N) then ...
      [] 'en'(NewPos) then
         {Browse 'Ascensor '#Num#' en el piso '#NewPos}
         case Plan
         of S|Plan2 then
            if NewPos==S then
               {Wait {Send Pisos.S llegada($)}}
               if Plan2==nil then
                  estado(NewPos nil false)
               else
                  {Send IdC paso(Plan2.1)}
                  estado(NewPos Plan2 true)
               end
            else ...
            end
         end
      end
   end
```

El Ascensor: Función de transición de estados (3)

```
fun {AscensorDE estado(Pos Plan EnMov) Msj}
      case Msj
      of piso(N) then ...
      [] 'en'(NewPos) then
         {Browse 'Ascensor '#Num#' en el piso '#NewPos}
         case Plan
         of S|Plan2 then
            if NewPos==S then
             ...
            else
               {Send IdC paso(S)}
               estado(NewPos Plan EnMov)
            end
         end
      end
   end
```

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Pruebas

- Observemos el flujo de control en los pisos, ascensores, controladores, y temporizadores.
- Por ejemplo, supongamos que hay diez pisos y dos ascensores. Ambos ascensores se encuentran en el piso 1, y solicitan un ascensor en el piso 9 y en el piso 10. ¿Cuáles son las ejecuciones posibles del sistema?

La edificación

Definamos un componente compuesto que crea una edificación con `NP` pisos y `NA` ascensores:

```
proc {Edificación NP NA ?Pisos ?Ascensores}
   Ascensores={MakeTuple ascensores NA}
   for I in 1..NA do IdC in
      IdC={Controlador estado(detenido 1 Ascensores.I)}
      Ascensores.I={Ascensor I estado(1 nil false) IdC Pisos}
   end
   Pisos={MakeTuple pisos NP}
   for I in 1..NP do
      Pisos.I={Piso I estado(nosolicitado) Ascensores}
   end
end
```

Ejecución de la edificación

```
declare P A in
{Edificación 10 2 P A}
{Send P.9 llamado}
{Send P.10 llamado}
{Send A.1 piso(4)}
{Send A.2 piso(5)}
```