



— *Imprimir\_Trabajo manda el trabajo a una impresora.*  
 — *No retorna hasta que el trabajo se ha terminado de imprimir.*  
**procedure** Imprimir\_Trabajo (Imp : **in** Rango\_Impresora;  
                                   Trabajo : **in** Tipo\_Trabajo);

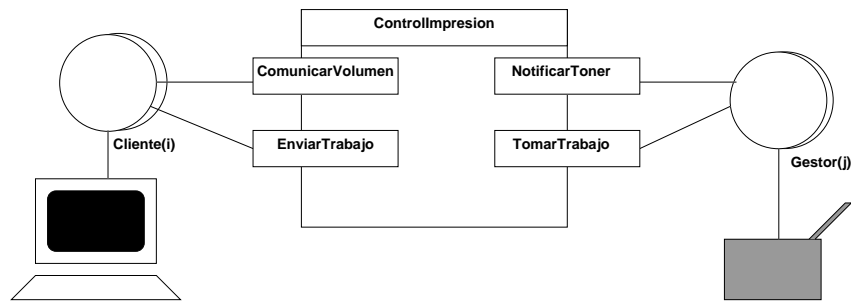
— *Pedir\_Toner notifica a la impresora que hay que cambiar el conjunto actual.*  
 — *No retorna hasta que se ha colocado el t  ner nuevo.*  
**procedure** Pedir\_Toner (Imp : **in** Rango\_Impresora);

**end** Impresora;

Adem  s, se asume la existencia de una operaci  n para detectar los trabajos de impresi  n seg  n se van generando:

— *Obtener\_Trabajo se ejecuta en cada ordenador y espera a que se produzca un trabajo de impresi  n.*  
 — *En Vol se devuelve tambi  n el tama  o de dicho trabajo.*  
**procedure** Obtener\_Trabajo (Trab : **out** TipoTrabajo;  
                                   Vol : **out** TipoVolumen);

Tras el an  lisis del problema se ha llegado al dise  o de procesos/recursos que se muestra en la figura:



El c  digo de las tareas Cliente es el siguiente:

**loop** — *Cliente*  
     ObtenerTrabajo(Trab , Vol);  
     Controlador.ComunicarVolumen(Vol);  
     Controlador.Env  iarTrabajo(Trab , Vol);  
**end loop**;

Tras refinar la interacción entre clientes y gestores llegamos a la siguiente especificación formal del recurso *ControlImpresion* — observad que faltan algunas partes.

### C-TADSOL *ControlImpresion*

#### OPERACIONES

**ACCIÓN** *ComunicarVolumen*:  $\text{ControlImpresion}[es] \times \text{Tipo\_Volumen}[e]$

**ACCIÓN** *EnviarTrabajo*:  $\text{ControlImpresion}[es] \times \text{Tipo\_Trabajo}[e] \times \text{Tipo\_Volumen}[e]$

**ACCIÓN** *NotificarToner*:  $\text{ControlImpresion}[es] \times \text{RangoImpresora}[e] \times \text{Tipo\_Volumen}[e]$

**ACCIÓN** *TomarTrabajo*:  $\text{ControlImpresion}[es] \times \text{RangoImpresora}[e] \times \text{Tipo\_Trabajo}[s] \times \mathbb{B}[s]$

#### SEMÁNTICA

##### DOMINIO:

**TIPO:**  $\text{ControlImpresion} =$

$(\text{libres: RangoImpresora} \rightarrow \mathbb{B} \times \text{queda: RangoImpresora} \rightarrow \text{Tipo\_Volumen} \times$   
 $\text{trabajos: RangoImpresora} \rightarrow \text{Tipo\_Trabajo} \times \text{volumenes: Tipo\_Volumen} \rightarrow \mathbb{N})$

**INICIAL(r):**  $\forall i \in 1..P \bullet \neg r.\text{libres}(i) \wedge$   
 $\forall k \in \mathbb{N} \bullet r.\text{volumenes}(k) = 0$

**CPRE:** Cierto

**ComunicarVolumen(r, vol)**

**POST:**  $r^{sal} = r^{ent} \setminus$   
 $r^{sal}.\text{volumenes} = r^{ent}.\text{volumenes} \oplus \{\text{vol} \mapsto 1 + r^{ent}.\text{volumenes}(\text{vol})\}$

**CPRE:**  $\exists j \in 1..P \bullet r.\text{libres}(j) \wedge r.\text{queda}(j) \geq \text{vol}$

**EnviarTrabajo(r, trab, vol)**

**POST:** [completar]

**CPRE:** Cierto

**NotificarToner(r, j, vol)**

**POST:**  $r^{sal}.\text{queda} = r^{ent}.\text{queda} \oplus \{j \mapsto \text{vol}\} \wedge$   
 $r^{sal}.\text{libres} = r^{ent}.\text{libres} \oplus \{j \mapsto \text{Cierto}\} \wedge$   
 $r^{sal}.\text{trabajos} = r^{ent}.\text{trabajos} \wedge$   
 $r^{sal}.\text{volumenes} = r^{ent}.\text{volumenes}$

**CPRE:** *Se dan las condiciones para mandar un trabajo a la impresora j o bien para decidir cambiar el tóner de dicha impresora.*

**CPRE:** [completar]

**TomarTrabajo(r, j, trab, excepcionToner)**

**POST:**  $(\neg(r^{ent}.\text{libres}(j)) \rightarrow \text{trab} = r^{ent}.\text{trabajos}(j) \wedge \text{excepcionToner} = \text{Falso} \wedge$   
 $r^{sal}.\text{trabajos} = j \triangleleft r^{ent}.\text{trabajos} \wedge$   
 $r^{sal}.\text{libres} = r^{ent}.\text{libres} \oplus \{j \mapsto \text{Falso}\} \wedge$   
 $r^{sal}.\text{volumenes} = r^{ent}.\text{volumenes})$   
 $\wedge$   
 $(r^{ent}.\text{libres}(j) \rightarrow \text{excepcionToner} = \text{Cierto} \wedge$   
 $r^{sal} = r^{ent} \setminus r^{ent}.\text{libres} \oplus \{j \mapsto \text{Falso}\})$



# Examen de Programación Concurrente

septiembre 2005

Dpto. LSIS. Unidad de Programación.

## Normas

En este examen se entregan tres partes diferenciadas: un planteamiento general, un cuestionario, y una petición de implementación. De ellas se entregan para su evaluación la segunda y la tercera. El cuestionario, que se entrega al principio del examen, debe rellenarse **en la misma hoja** en la que se entrega (consulta a un profesor si necesitas usar espacio adicional). Se recogerá **una hora y media** tras haberse entregado.

La puntuación total del examen es de **10 puntos**.

Las soluciones se proporcionarán antes de la revisión y las calificaciones se darán a conocer el **20 de septiembre de 2005**. La revisión de este examen tendrá lugar el **21 de septiembre de 2005**.

Apellidos:\_\_\_\_\_

Nombre:\_\_\_\_\_ Número de matrícula:\_\_\_\_\_

## 1. Cuestionario [4 puntos]

### 1.1. Completar tarea [1 punto]

Completar aquí el pseudocódigo correspondiente a la tarea Gestor(j).

### 1.2. Estudiar el comportamiento [1 punto]

Con este diseño, ¿puedes imaginar algún escenario en el que hay trabajos para imprimir, ninguno se imprime y tampoco se da ningún mensaje en las impresoras?

Justifica la respuesta.

**1.3. Completar especificación**

**[1 punto]**

Completar aquí la especificación de la postcondición de EnviarTrabajo.

**1.4. Completar especificación**

**[1 punto]**

Completar aquí la especificación de la precondición de TomarTrabajo.

# Examen de Programación Concurrente

septiembre 2005

Dpto. LSIIS. Unidad de Programación.

## Normas

Disponéis de **una hora y media** para responder a los dos apartados de esta parte.  
Cada uno de ellos se deberá entregar en un conjunto de hojas separadas.

## 2. Implementación

[6 puntos]

Para cada uno de los apartados siguientes ha de entregarse lo que se pide en ellos. Recordad **entregar la respuesta a cada apartado en un juego de hojas separadas**.

Se asumirá que la especificación del sistema ha sido completada de la siguiente manera:

El código de las tareas Gestor es el siguiente:

```
loop — Gestor(J)
  Impresora.Estimar_Restante(J, Queda);
  Controlador.NotificarToner(J, Queda);
  Controlador.TomarTrabajo(J, Trab, Excepcion);
  if Excepcion then
    Impresora.Pedir_Toner(J);
  else
    Impresora.Imprimir_Trabajo(J, Trab);
  end if;
end loop;
```

Y las operaciones del C-TADSOL quedan como sigue:

**CPRE:**  $\exists j \in 1..P \bullet r.libres(j) \wedge r.queda(j) \geq vol$

**EnviarTrabajo(r, trab, vol)**

**POST:**  $\exists j \in 1..P \bullet (r^{ent}.libres(j) \wedge r^{ent}.queda(j) \geq vol \wedge$   
 $r^{sal}.trabajos = r^{ent}.trabajos \oplus \{j \mapsto trab\} \wedge$   
 $r^{sal}.libres = r^{ent}.libres \oplus \{j \mapsto Falso\}) \wedge$   
 $r^{sal}.queda = r^{ent}.queda \wedge$   
 $r^{sal}.volumenes = r^{ent}.volumenes \oplus \{vol \mapsto r^{ent}.volumenes(vol) - 1\}$

**CPRE:**  $\neg(r.libres(n)) \vee$

$(r.queda(n) < MIN \wedge \exists v \bullet r.volumenes(v) > 0 \wedge v > \max_{i=1}^P r.queda(i)) \wedge$

$\neg \exists v < r.queda(j) \bullet r.volumenes(v) > 0$

**TomarTrabajo(r, j, trab, excepcionToner)**

**POST:**  $(\neg(r^{ent}.libres(j)) \rightarrow trab = r^{ent}.trabajos(j) \wedge excepcionToner = Falso \wedge$   
 $r^{sal}.trabajos = j \triangleleft r^{ent}.trabajos \wedge$   
 $r^{sal}.libres = r^{ent}.libres \oplus \{j \mapsto Falso\} \wedge$   
 $r^{sal}.volumenes = r^{ent}.volumenes)$   
 $\wedge$   
 $(r^{ent}.libres(j) \rightarrow excepcionToner = Cierto \wedge$   
 $r^{sal} = r^{ent} \setminus r^{ent}.libres \oplus \{j \mapsto Falso\})$

**2.1. Objetos protegidos****[3 puntos]**

Realizar una implementación en Ada 95 de las operaciones `EnviarTrabajo` y `TomarTrabajo` del **C-TADSOL** utilizando objetos protegidos y que incluya la definición de tipos y variables de estado necesarias.

**2.2. *Rendez-Vous* / Paso de mensajes****[3 puntos]**

Realizar una implementación en Ada 95 de las operaciones `EnviarTrabajo` y `TomarTrabajo` del **C-TADSOL** utilizando *Rendez-Vous* (con paso de mensajes explícito si se considera necesario, modificando en este caso el código del cliente para hacer explícito el envío / recepción a través de los canales) y que incluya la definición de tipos y variables de estado necesarias y el código de todas las operaciones solicitadas.