



# Sistemas Inteligentes en Tiempo Real

Vicente J. Botti Navarro  
Departamento de Sistemas Informáticos y Computación  
Universidad Politécnica de Valencia

1

## Indice



- Objetivo
- Conceptos Generales
  - Sistemas de Tiempo Real e Inteligencia Artificial en Tiempo Real
  - Requerimientos de un SIATR / ATR
  - Técnicas y arquitecturas
  - Limitaciones
  - Agentes / Sistemas Multiagente
- ARTIS
  - Modelo de Entorno
  - Arquitectura
  - Prototipo
  - Extensiones

2

# Objetivos de un Agente Inteligente en Tiempo Real



3

# Sistemas de Tiempo Real

- STR: sistema informático cuyo comportamiento correcto depende de la validez lógica de los resultados y del instante en que se producen
- Requisitos temporales: plazo, periodo, precedencia...
- Clasificación de tareas: críticas, acríicas, periódicas, aperiódicas y esporádicas.
- Planificación de STR: asignar recursos a las tareas de forma que cumplan sus requisitos temporales.
- Métricas: altas utilizaciones de los recursos, producir un bajo aumento de la sobrecarga, asegurar la estabilidad, existencia de productos comerciales, predecibilidad
- Predecibilidad ==> análisis de la planificabilidad.

4

# Paradigmas de Planificación de STR

- Aproximaciones estáticas basadas en tablas
  - las técnicas de los ejecutivos cíclicos (Baker, 1989)
  - los trabajos de Xu y Parnas (Xu, 1990).
- Aproximaciones estáticas dirigidas por prioridades expulsivas
  - asignación estática al más frecuente ó RM (Rate Monotonic)
  - asignación estática al más urgente ó DM (Deadline Monotonic)
  - asignación dinámica al más urgente ó EDF (Earliest Deadline First)
  - asignación dinámica al más holgado ó LLF (Least Laxity First)
- Aproximaciones dinámicas basadas en planes
  - trabajos realizados en el sistema SPRING (U. de Massachusets)
- Aproximaciones dinámicas del mejor resultado (best-effort)
  - los trabajos de Locke (Locke, 1986)

5

## IATR

- ❖ Creciente interés en la aplicación de técnicas de IA para la resolución de problemas en entornos complejos, dinámicos, e incompletamente especificados.
- ❖ Convergencia en el desarrollo de dos de las áreas de investigación en Informática:

S.T.R.



I.A.

• Crear sistemas más sofisticados  
==> aplicación de técnicas de IA

• Utilizar los sistemas inteligentes en entornos de trabajo en tiempo real ==> dotarlos de predecibilidad

6

## Objetivo en IATR

❖ Combinar y potenciar las ventajas de ambas áreas de investigación, la flexibilidad de las técnicas de IA, con la predecibilidad de los métodos utilizados en el desarrollo de STR, para poder construir aplicaciones de tiempo real estricto con componentes inteligentes.

- ◆ análisis de modelos de tareas y técnicas de planificación de STR
- ◆ determinar los requerimientos de los SIATR, y análisis de las diversas técnicas y arquitecturas propuestas en la literatura

◆ Definición de un modelo de entidades (usuario + sistema) y una arquitectura software (ARTIS: an Architecture for Real Time Intelligent Systems) que organice y planifique las acciones a realizar.

◆ Desarrollo, extensión, modificación o adaptación de los métodos existentes a la arquitectura propuesta.

7

## Requerimientos de un SIATR

- ➔ ■ Operar de forma continua sobre extensos periodos de tiempo.
- ➔ ■ Interrelacionarse con el entorno externo a través de sensores y actuadores.
  - Tratar con datos inciertos e incompletos.
  - Concentrar recursos hacia los eventos más críticos.
- ➔ ■ Tratar tanto los eventos síncronos como asíncronos de una forma predecible con tiempos de respuesta garantizados.
- ➔ ■ Degradarse gradualmente.
  - Representación y razonamiento temporal
  - No monotonía.

8



# Técnicas y Arquitecturas de IATR

- Poda
- Ordenación
- Aproximación
- Limitación del alcance

- Técnicas básicas:

- Razonamiento progresivo (Winston)
- Algoritmos anytime (Boddy)
- Computaciones imprecisas (Liu)
- Búsqueda en tiempo real (Korf)
- Métodos múltiples (Lesser)

9



# Técnicas y Arquitecturas de IATR

- Arquitecturas:

- AIS (Adaptative Intelligent System)  
(Hayes-Roth)
- CIRCA (The Cooperative Intelligent Real-  
Time Control Architecture) (Musliner)
- Phoenix (Howe)
- PRS (Ingrand)
- REAKT

10

# Técnicas y Arquitecturas de IATR

## ❖ Sistemas de IA integrados en un STR

- ◆ arquitecturas IA deliberativas
- ◆ técnicas de aproximación
- ◆ métodos múltiples
- ◆ arquitecturas IA puramente reactivas
- ◆ algoritmos any-time
- ◆ sistemas de planificación deliberativa



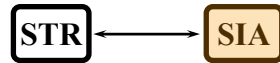
## ❖ Integración del tiempo real en IA

- ◆ modificación de los sistemas de producción
- ◆ sistemas basados en pizarras interrumpibles



## ❖ Cooperación de SIA y de TR

- ◆ arquitectura CIRCA



11

# Limitaciones de las técnicas y arquitecturas de IATR

ARTIS

## ■ Arquitecturas:

- AIS, PRS, Phoenix, ABE/RT
  - no garantía de tiempo de respuesta, no permiten tiempo real estricto.
- CIRCA
  - no reacciona ante eventos no previstos, planificación poco flexible (ejecutivo cíclico), poca reactividad del componente de IA
- REAKT
  - no predecibilidad modelo de agentes, carencia extensiones RTOS (procesos pesados)

12

# Limitaciones de las técnicas y arquitecturas de IATP

MKS

## ■ Técnicas:

- Algoritmos anytime
  - dificultad diseño (monotonía), semántica perfiles de ejecución
- Métodos múltiples
  - baja utilización, poca reactividad, dificultad validación, no mejora con el tiempo
- Razonamiento progresivo, búsqueda en tiempo real
  - no predecibilidad, no tiempo real estricto (no garantía solución)
- Computaciones imprecisas
  - no posibilidad de mejorar solución, degradación brusca

13

# Agente / Sistemas Multiagente

- Un agente es todo aquello que puede considerarse que percibe su entorno mediante sensores y actúa sobre tal entorno por medio de actuadores (Russell).
- Un agente racional ideal es aquel que en todos los casos de posibles secuencias de percepciones deberá emprender todas aquellas acciones que favorezcan obtener el máximo de su función de ejecución, basándose en las evidencias aportadas por la secuencia de percepciones y en el conocimiento incorporado en el agente (Russell).
- Un agente es un sistema informático, situado en algún entorno, que percibe el entorno (entradas sensibles de su entorno) y a partir de tales percepciones determina (mediante técnicas de resolución de problemas) y ejecuta acciones (de forma autónoma y flexible) que le permiten alcanzar sus objetivos y que pueden cambiar el entorno (Wooldridge).
- Los agentes son componentes persistentes activos que perciben, razonan, actúan y se comunican (Huns).

14



## Agente / Sistemas Multiagente

---

- Propiedades de un agente (Franklin, Gaesser, Nwana, ....):
  - Continuidad temporal
  - Autonomía
  - Sociabilidad
  - Racionalidad
  - Reactividad
  - Proactividad
  - Adaptabilidad
  - Movilidad
  - Veracidad
  - Benevolencia

15




## Requerimientos de un RT Agente

---

- Garantía del 100% del tiempo de respuesta para acciones críticas.
- Percepción del entorno por medio de sensores y actuación sobre el mismo a través de actuadores (reactivo).
- Obtención de respuestas mediante un proceso reflejo o deliberativo.
- Restricciones estrictas en entornos dinámicos.
- Debe poder ser integrado en un sistema multiagente.

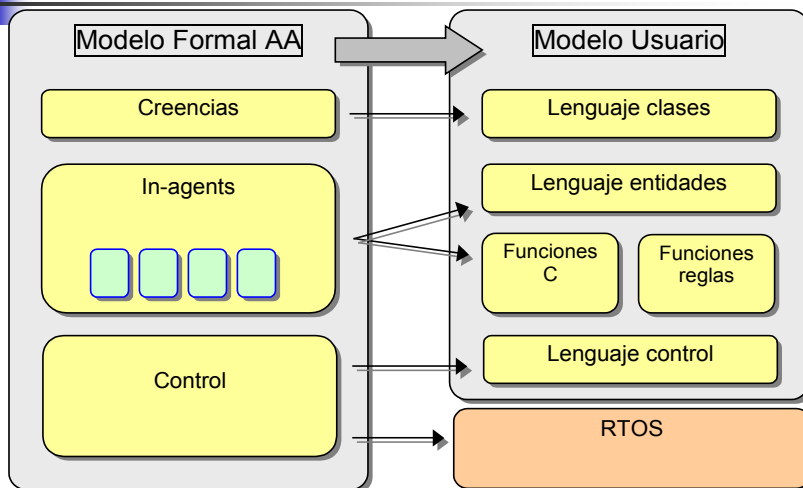
16

# ARTIS

- ARTIS
- 
- Objetivo
  - Aportaciones
  - Modelo de Entorno
  - Arquitectura
  - Modelo asignación prioridades
  - Prototipo
  - Conclusiones
  - Extensiones

17

## ARTIS / Objetivo



18

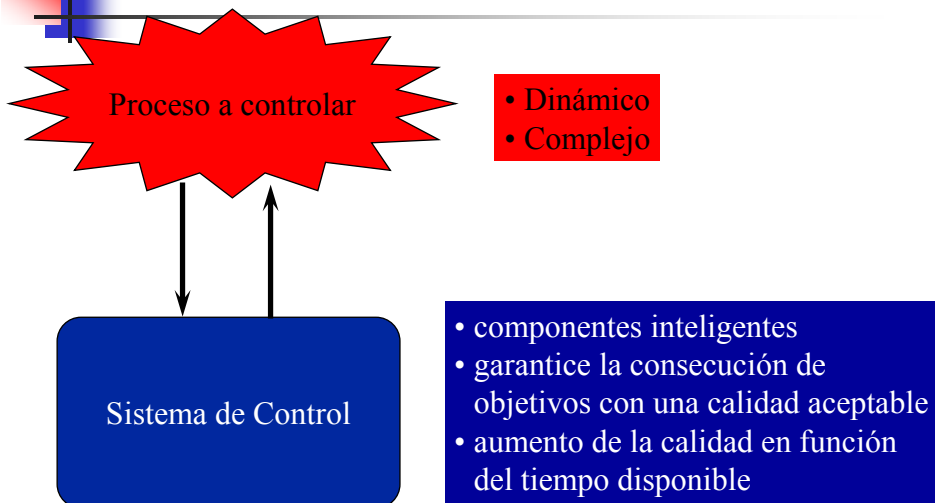
# ARTIS Objetivo

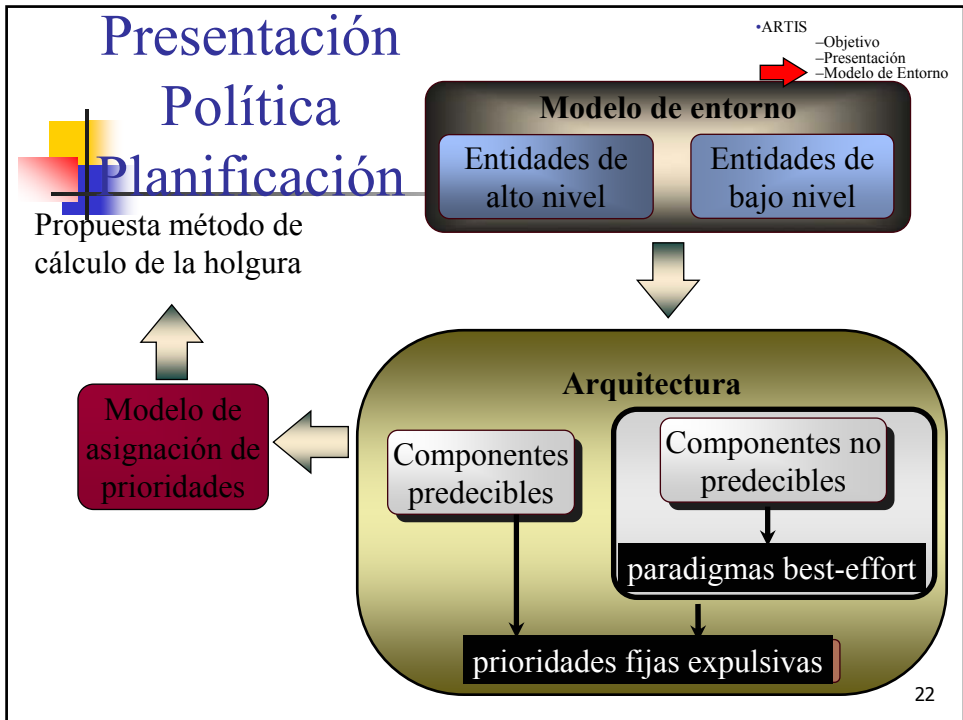
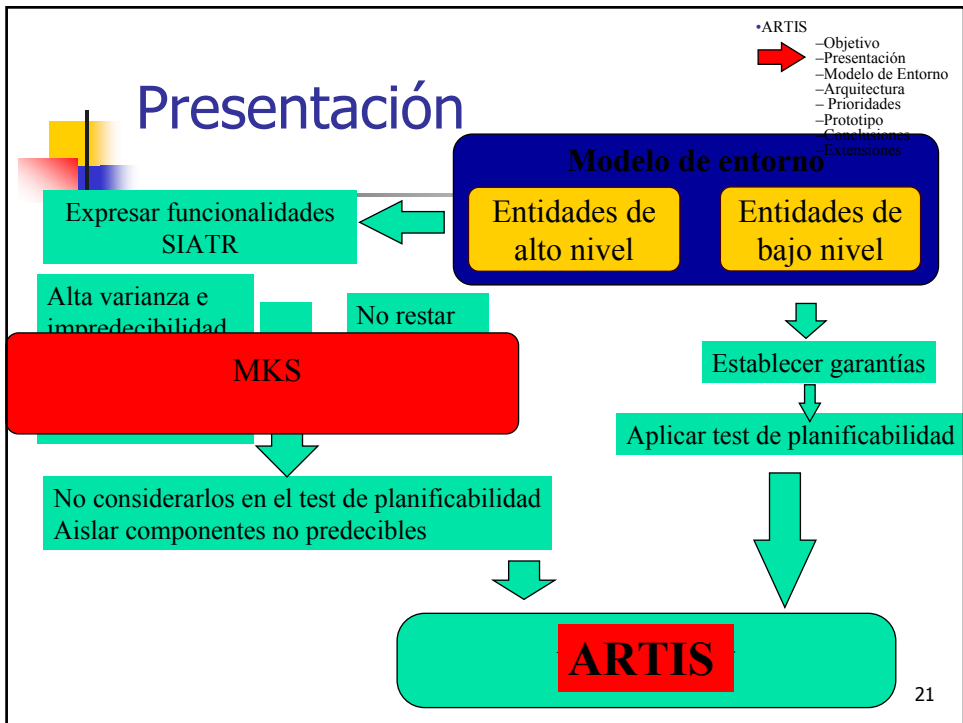
## Técnica de IATR

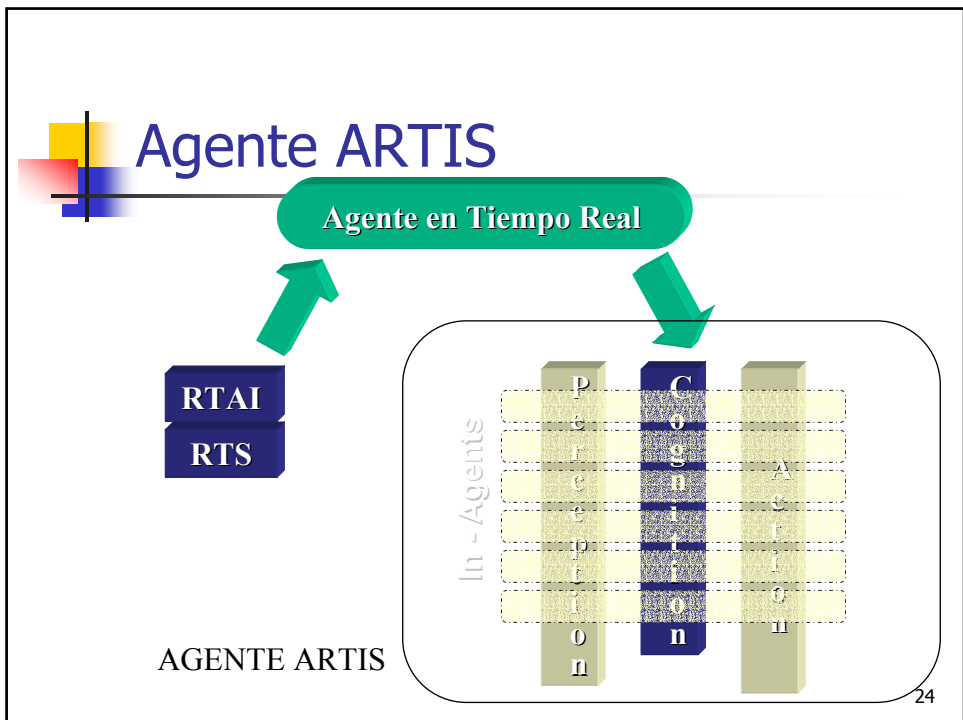
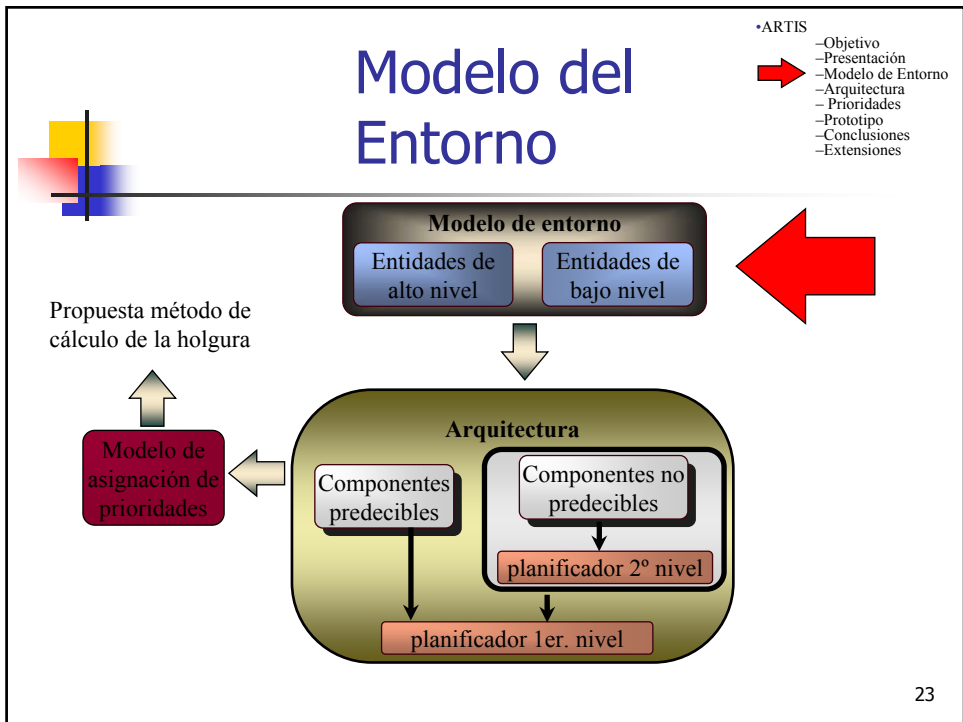
- predecibilidad
- alta reactividad
- fácil de diseñar y validar
- pueda ser utilizada en tiempo real estricto
- degradación gradual
- guiado por la utilidad
- **Arquitectura**
  - garantía tiempo de respuesta para tiempo real estricto
  - análisis de planificabilidad
  - alta reactividad
  - guiado por la utilidad



# Presentación General

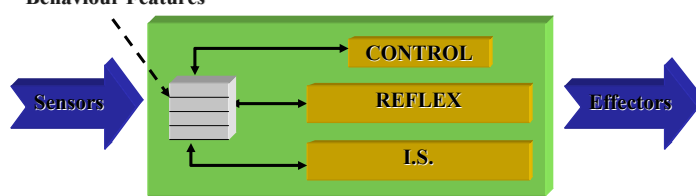






# Propiedades Agente ARTIS

- Autonomía, Reactividad, Proactividad y Continuidad Temporal.
- Puede incluir más características.
- AA básico diseñado para actuar correctamente en tiempo real estricto.
- Algunas de las características adicionales pueden entrar en contradicción con la conducta en tiempo real estricto debido a la impredecibilidad de las acciones que implican.



25

## Modelo de Entidades de Alto Nivel

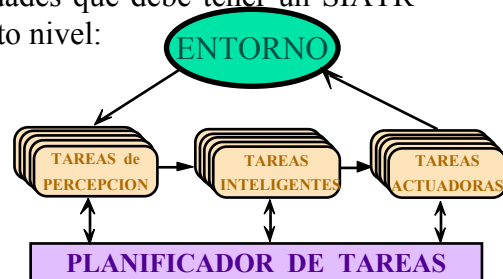
- \*ARTIS
  - Objetivo
  - Presentación
  - Modelo de Entorno
  - Arquitectura
  - Prioridades
  - Prototipo
  - Conclusiones
  - Extensiones

- Los objetivos subyacentes en el modelo de entidades propuesto son: **Multi-Knowledge- Source**

- resolver la necesidad de una reacción del sistema, cuando se produce un cambio en el problema, en un intervalo de tiempo finito y predefinido,
- proporcionar las funcionalidades que debe tener un SIATR desde el punto de vista de alto nivel:

- Percepción
- Cognición
- Acción

**In-Agent**



# Formalización AA

- Constituida:
  - Un conjunto de **Módulos de Percepcion** y **Action**.
  - **Módulo de Control** :
    - Ejecución en tiempo real de cada componente.
    - Algunas características de comportamiento:
      - Implementan diferentes atributos dependiendo de la clase de problema a resolver.
  - **Modulo Reflejo**.
  - **Servidor Inteligente (IS)**.

27

# Formalización AA

- $AA = (U_{AIA}, f_{AIA}, B)$   
donde:
  - $U_{AIA}$  = un conjunto de agentes internos (in-agents).
  - $f_{AIA}$  **Una función de selección de agentes internos. Implementada por el Módulo de Control junto con el Servidor Inteligente.**
  - $B$  = **Un conjunto de creencias que representan el entorno y los estados internos del agente. Consituido por las creencias de todos los in-agente y un conjunto global de creencias.**

28

# In-Agent

- ARTIS
- Objetivo
- Aportaciones
- Modelo de Entorno
- Arquitectura
- Prioridades
- Prototipo
- Conclusiones
- Extensiones

Entidad Interna con el conocimiento necesario para resolver un problema concreto (Es un conjunto de MKS's que cooperan para resolver un subproblema del problema global).

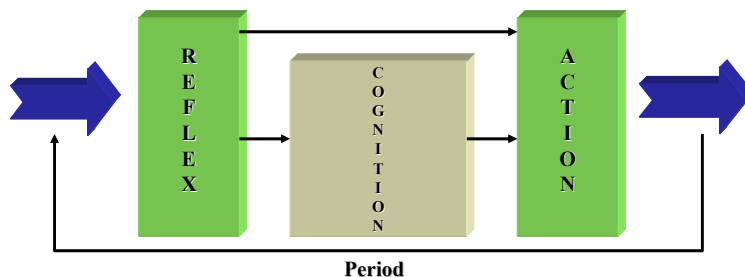
- El usuario puede definir un plazo máximo de ejecución y un tiempo mínimo de llegada entre la ocurrencia de eventos o periodo asociados a cada actividad.
- Su estructura conceptual consta de tres componentes:



29

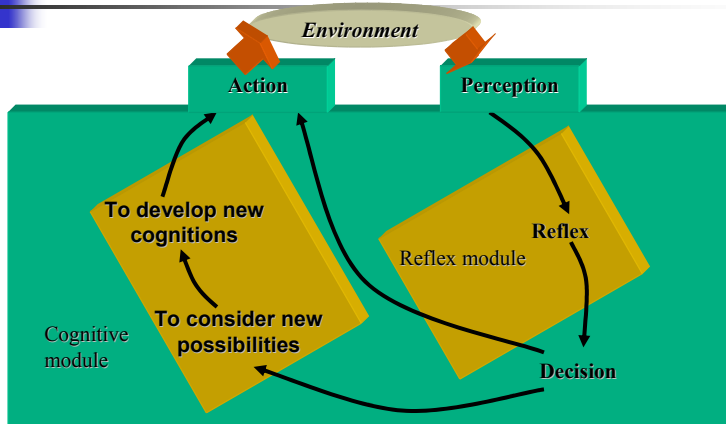
# Comportamiento In-Agent

- Nivel Reflejo.
- Nivel de Acción.
- Nivel Cognitivo.



30

## Ciclo de vida de un In-Agent



31

## Formalización In-Agent

$$\text{IN-AGENT} = (B_{AIA}, L_r, f_r, L_c, f_c, D, T)$$

donde:

- $B_{AIA}$  = Un conjunto de creencias que representan el estado interno y el entorno del in-agent.
  - La información tiene una ventada de validez temporal.
- $L_r$  = Lista de todas las posibles acciones reflejas conocidas por el in-agent.
- $f_r$  = Función de selección de una acción refleja de  $L_r$ .
- $L_c$  = Lista de todas las posibles acciones cognitivas conocidas por el in-agent.
- $f_c$  = Función de selección de una acción refleja de  $L_c$ .
- $D$  = Plazo máximo de ejecución.
- $T$  = Periodo.

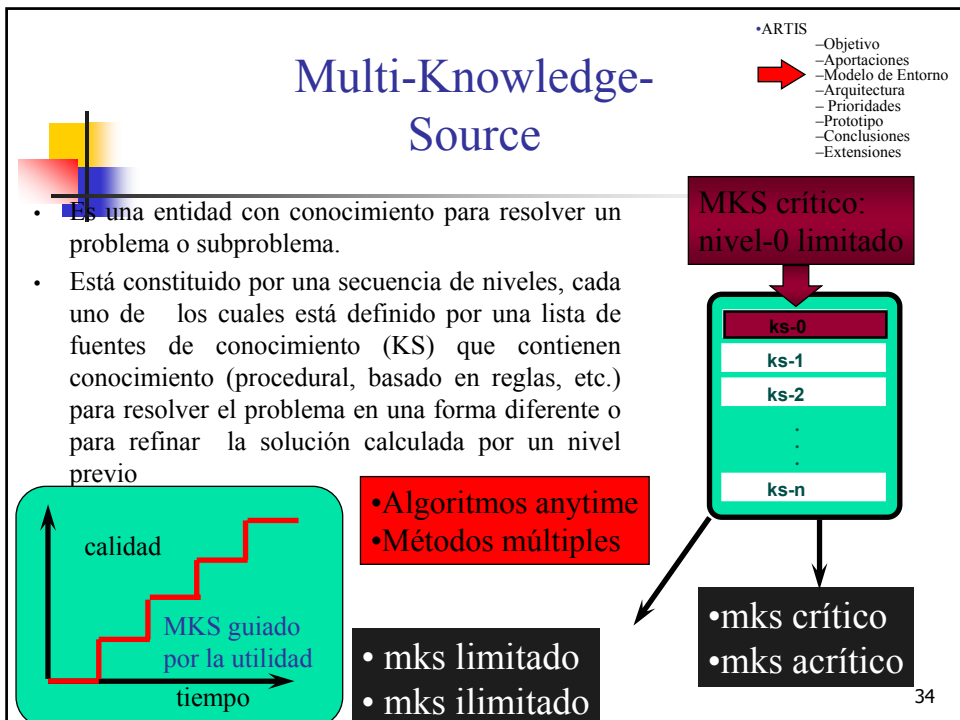
32

# Interprete In-Agent

```

Perception(t) ← Read(Environment, t)
BAIA ← BAIA ∪ Perception(t)
(BAIA, Reflex_Action) ← fr(BAIA, Lr)
(BAIA, Cognition_Action) ← fc(BAIA, Lc, Reflex_Action, D)
if (Cognition_Action = not_completed)
    Execute(Reflex_Action)
else Execute(Cognition_Action)
T
  
```

33

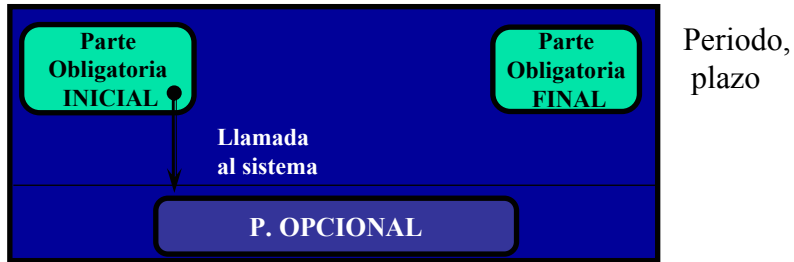


34

# Modelo de Entidades de Bajo Nivel

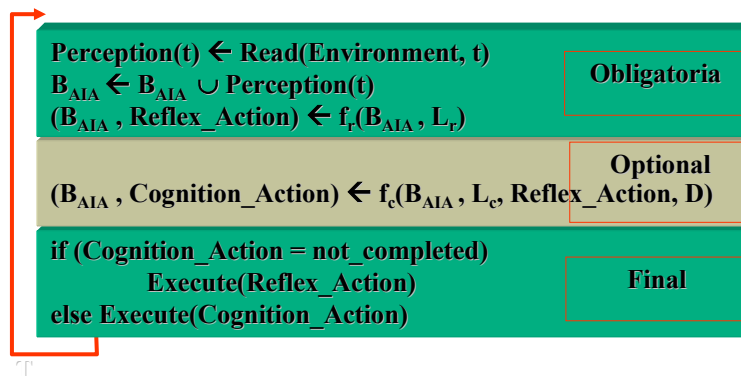
- \*ARTIS
- Objetivo
- Aportaciones
- Modelo de Entorno
- Arquitectura
- Prioridades
- Prototipo
- Conclusiones
- Extensiones

❖ Cada agente del modelo anterior debe ser traducido a un modelo de entidades a ser planificadas por el sistema operativo. A dicha entidad la denominaremos tarea de bajo nivel o simplemente tarea.



Estructura conceptual de una tarea de bajo nivel

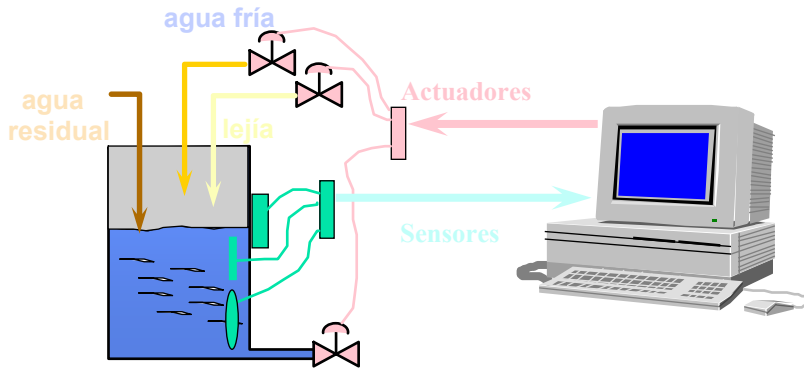
# Interprete In-Agent



T

# Ejemplo

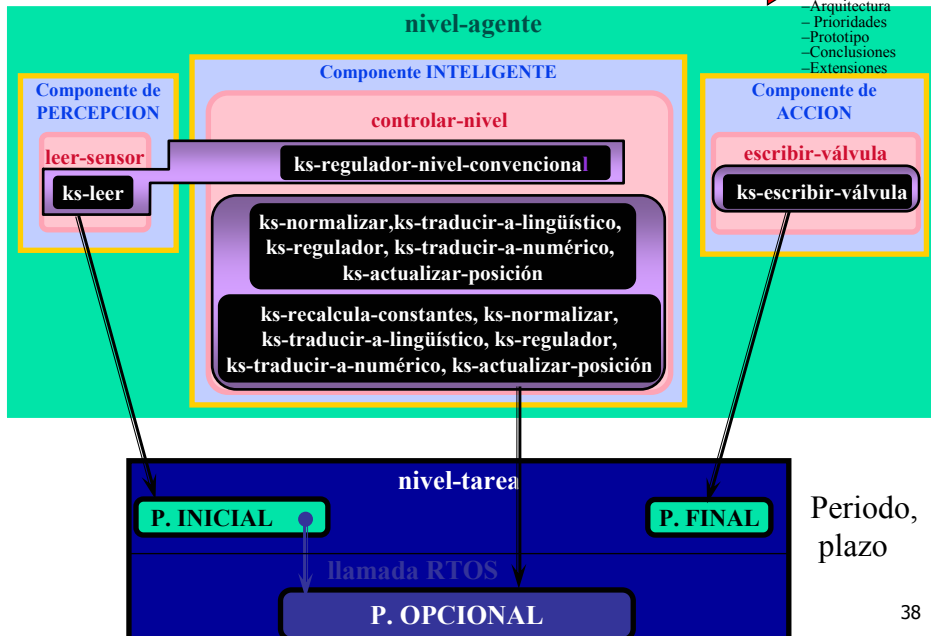
- \*ARTIS
- Objetivo
- Aportaciones
- Modelo de Entorno
- Arquitectura
- Prioridades
- Prototipo
- Conclusiones
- Extensiones



37

# Ejemplo

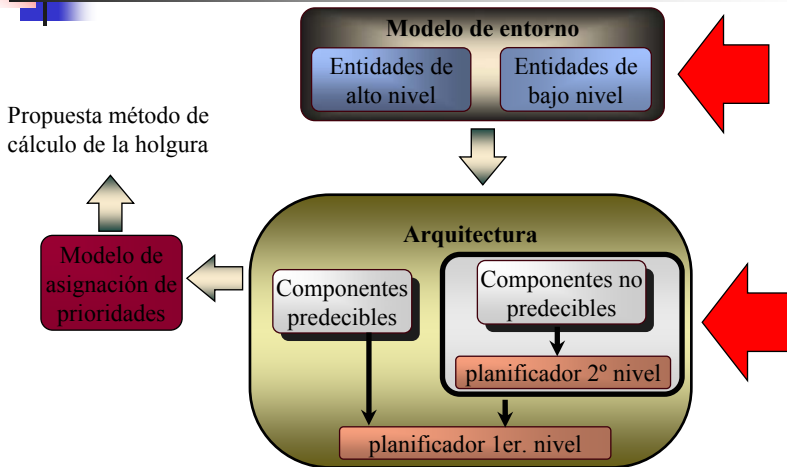
- \*ARTIS
- Objetivo
- Aportaciones
- Modelo de Entorno
- Arquitectura
- Prioridades
- Prototipo
- Conclusiones
- Extensiones



38

# Arquitectura

- \*ARTIS
- Objetivo
- Aportaciones
- Modelo de Entorno
- Arquitectura
- Prioridades
- Prototipo
- Conclusiones
- Extensiones



39

# Política Planificación

- \*ARTIS
- Objetivo
- Aportaciones
- Modelo de Entorno
- Arquitectura
- Prioridades
- Prototipo
- Conclusiones
- Extensiones

- El paradigma de planificación escogido debe garantizar estáticamente el cumplimiento de todos los plazos críticos, a la vez que debe ser flexible y fácilmente adaptable a las nuevas situaciones que se producen en entornos cambiantes.
- Utilizar varios paradigmas simultáneamente, estableciendo distintos niveles de planificación:
  - En el nivel más bajo ==> utilizar paradigmas estáticos que garanticen a priori la viabilidad de las tareas
  - En niveles posteriores ==> utilizar tests de garantía on-line y paradigmas best-effort que mejoren la calidad de la respuesta según el tiempo disponible.

40

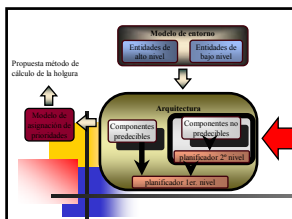
# Política Planificación

- \*ARTIS
- Objetivo
- Aportaciones
- Modelo de Entorno
- Arquitectura
- Prioridades
- Prototipo
- Conclusiones
- Extensiones



- Primer nivel de planificación ==> paradigma basado en la asignación de prioridades fijas expulsivas.
- Razones:
  - Este tipo de políticas cumplen con los objetivos propuestos superando los inconvenientes de las políticas basadas en tablas.
  - La teoría de planificación de asignación de prioridades fijas expulsivas, permite a los ingenieros de software razonar acerca de la corrección temporal de un conjunto de tareas a un alto nivel de abstracción.
  - La aplicación de esta teoría a las aplicaciones de tiempo real coloca el desarrollo y mantenimiento de los STR en una base de ingeniería analítica, haciendo esos sistemas fáciles de desarrollar y mantener.

41



# Componentes de ARTIS

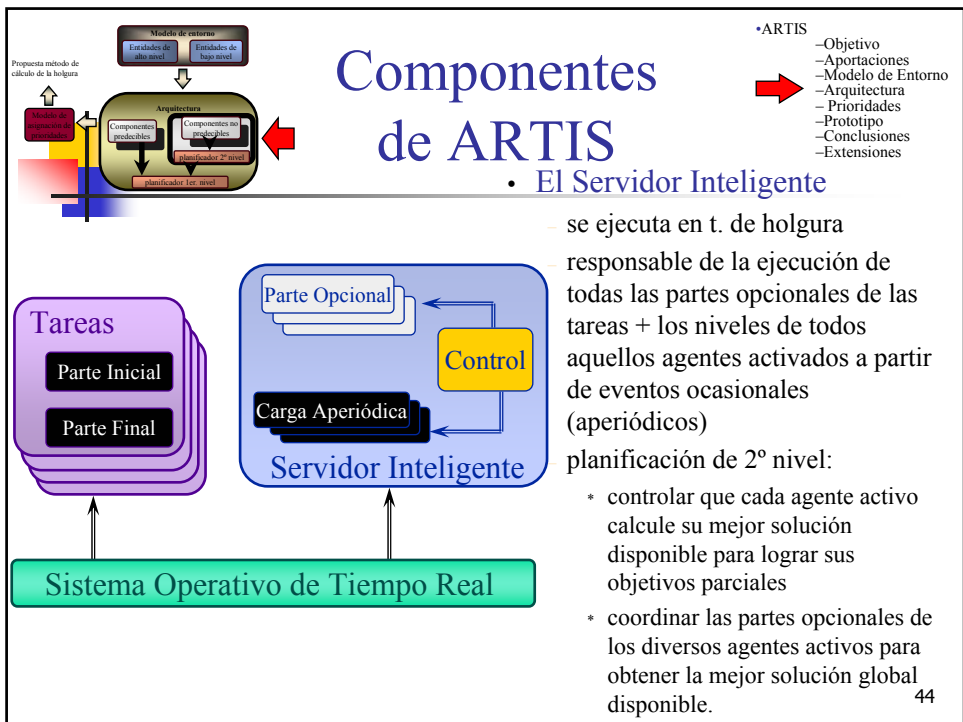
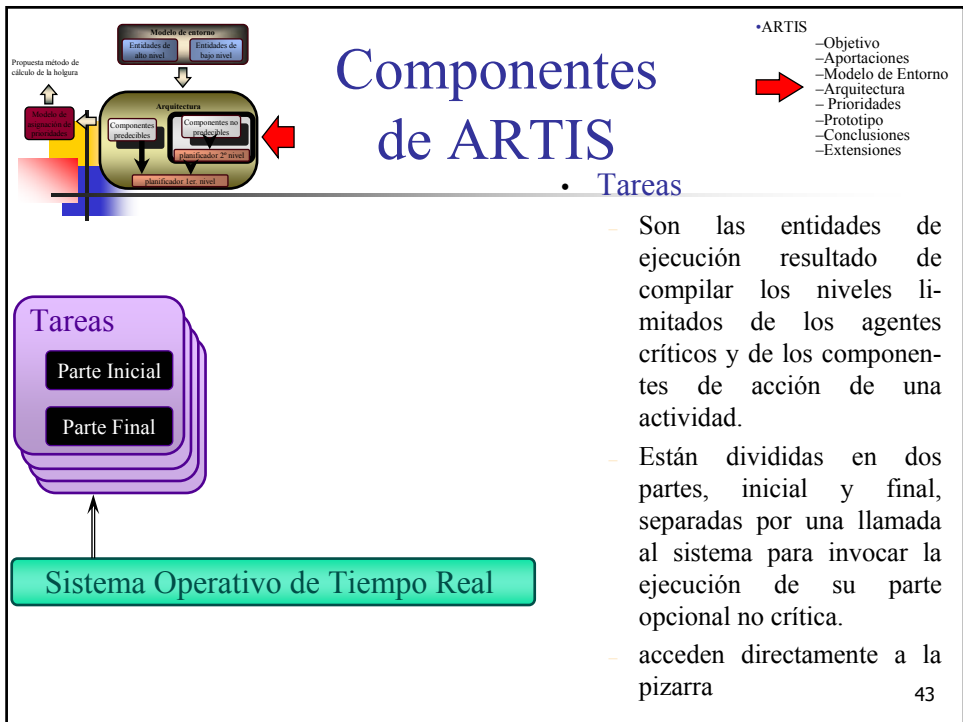
- \*ARTIS
- Objetivo
- Aportaciones
- Modelo de Entorno
- Arquitectura
- Prioridades
- Prototipo
- Conclusiones
- Extensiones

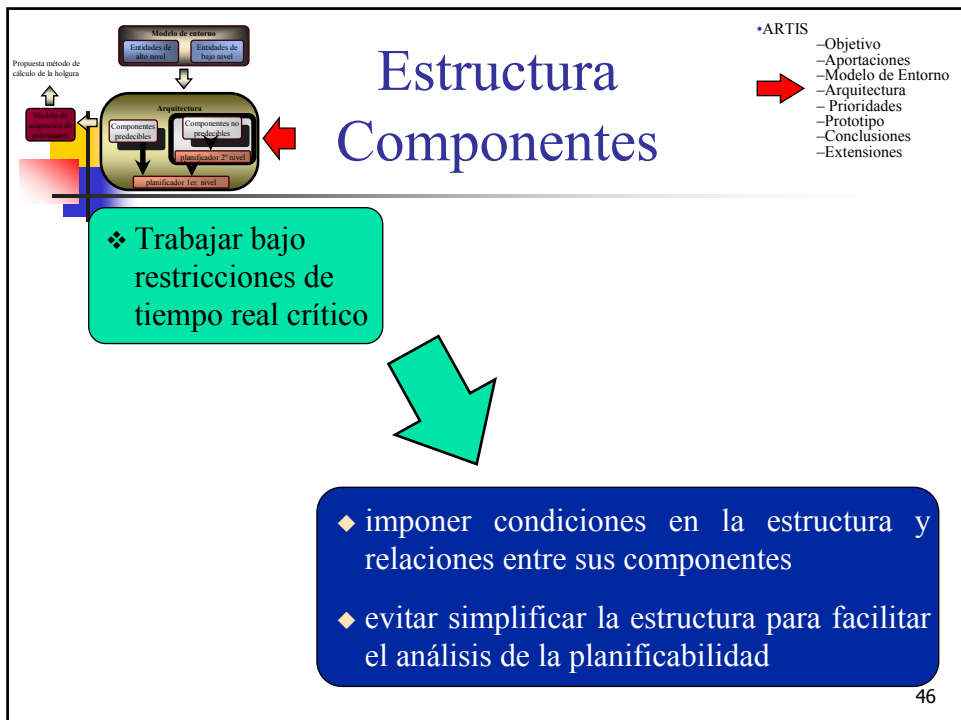
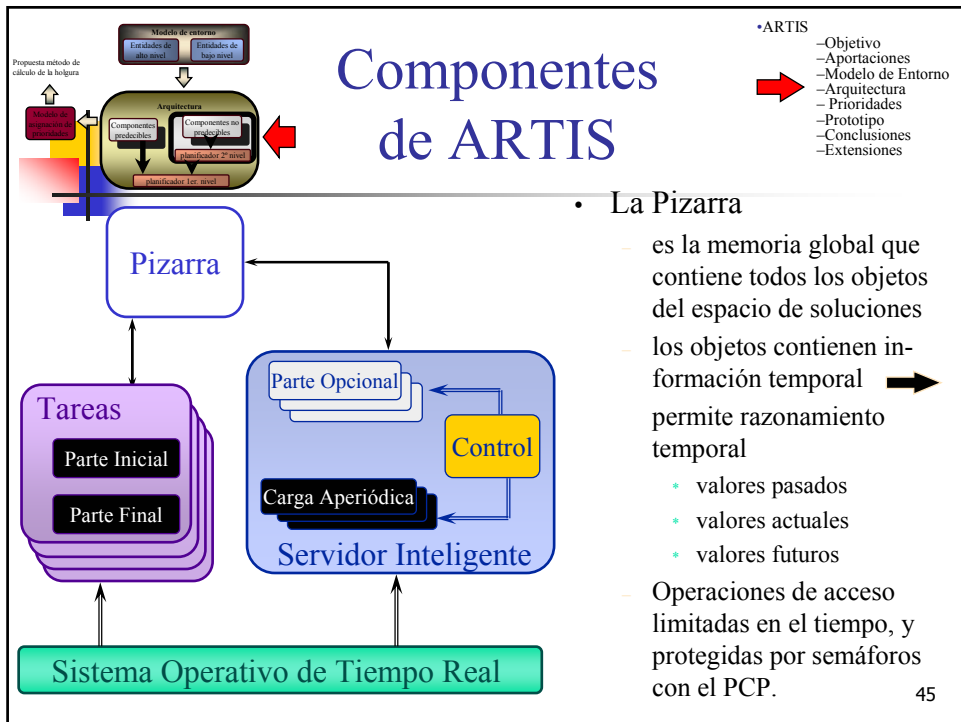


- El sistema operativo de tiempo real de base
  - activación de tareas periódicas
  - permitir varios niveles de planificación
  - planificación por prioridades fijas expulsivas
  - cálculo y mantenimiento de la holgura
  - compartición de memoria semáforos con PCP y colas de mensajes con prioridad
  - Control del wctet

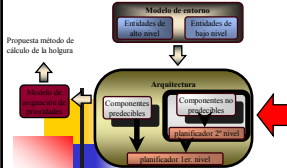
Sistema Operativo de Tiempo Real

42





# Estructura Componentes Opcionales



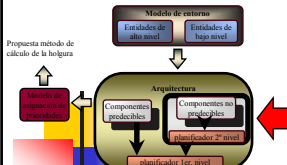
- \*ARTIS
  - Objetivo
  - Aportaciones
  - Modelo de Entorno
  - Arquitectura
  - Prioridades
  - Prototipo
  - Conclusiones
  - Extensiones

- los componentes opcionales trabajarán con su propia memoria local
- cola de mensajes
- su estructura está dividida en tres fases:
  - lectura, cálculo y commit
- sólo podrán acceder directamente a la pizarra en su fase de commit
- la fase de commit es atómica

```

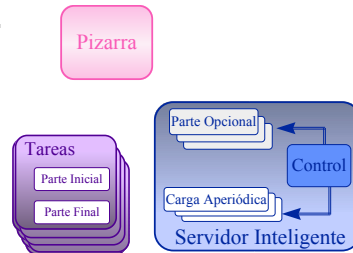
Inicialización;
mientras !fin_cálculos hacer
    leer_mensajes_pendientes();
    cálculos();
fmientras
mientras !commit_hecho hacer
    si puedo_hacer_commit()
    entonces
        escribir_cambios_en_pizarra();
        commit_hecho= TRUE;
    sino    avisar_al_control();
    fsi
fmientras
fin_KSI();
    
```

# Estructura Componentes Críticos



- \*ARTIS
  - Objetivo
  - Aportaciones
  - Modelo de Entorno
  - Arquitectura
  - Prioridades
  - Prototipo
  - Conclusiones
  - Extensiones

- las tareas críticas no informan de los cambios directamente a los componentes opcionales sino al control
- los accesos para escritura en la pizarra generarán los correspondientes eventos para el componente de control
- las ventajas de esta decisión son:
  - los mensajes se envían en tiempo de holgura, con lo que no afectan al umbral de la planificabilidad,
  - sólo es necesario enviar los mensajes a los componentes opcionales que estén activos y además puedan aprovecharlo.



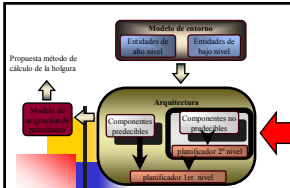
Sistema Operativo de Tiempo Real

```

cálculos_parte_inicial();
petición_ejecución_parte_opcional();
cálculos_parte_final();
fin_tarea();
    
```

# Estructura Módulo Control

- ARTIS
- Objetivo
- Aportaciones
- Modelo de Entorno
- Arquitectura
- Prioridades
- Prototipo
- Conclusiones
- Extensiones



- Fases de un ciclo de Control clásico
- cola de mensajes con acceso atómico
- se le informa de la holgura y el instante en que será interrumpido.
- se activa al inicio de su ciclo cada vez que ocurran nuevos eventos en el sistema.

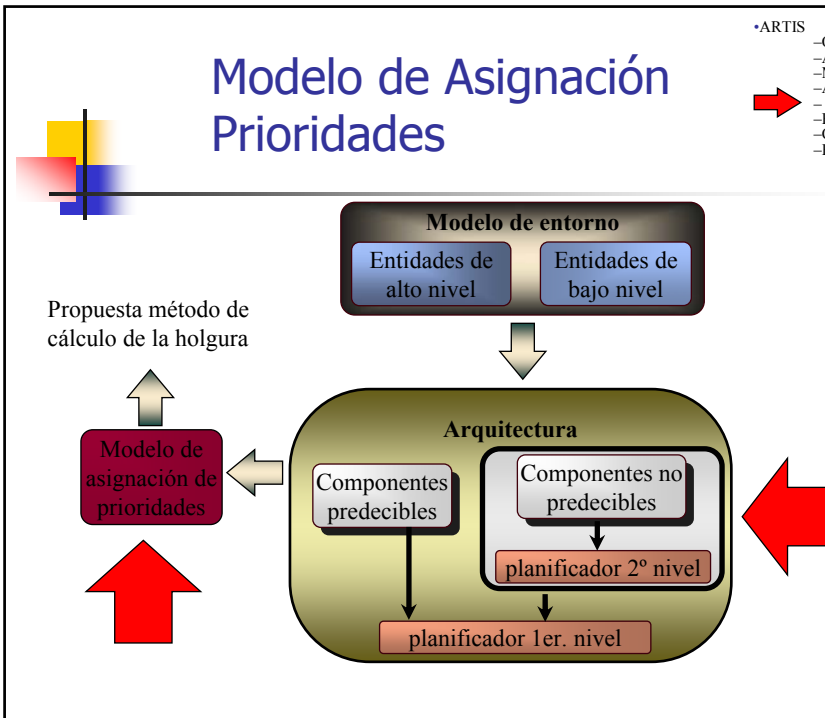
```

Inicialización();
para_siempre_hacer
si !suficiente_tiempo() entonces fin_control();
sino
leer_eventos_cola_mensajes();
si !suficiente_tiempo() entonces fin_control();
sino
comprobar_condiciones();
si !suficiente_tiempo() entonces fin_control();
sino
valoración();
si !suficiente_tiempo() entonces fin_control();
sino
seleccionar_KSI();
enviar_mensajes_KSI_seleccionado();
ejecutar_KSI_seleccionado();

fsi
fsi
fsi
fsi
fin_para
    
```

# Modelo de Asignación Prioridades

- ARTIS
- Objetivo
- Aportaciones
- Modelo de Entorno
- Arquitectura
- Prioridades
- Prototipo
- Conclusiones
- Extensiones



## Prioridades fijas expulsivas

\*ARTIS

- Objetivo
- Aportaciones
- Modelo de Entorno
- Arquitectura
- Prioridades
- Prototipo
- Conclusiones
- Extensiones

- **Análisis básico**  $r_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_j}{T_j} \right\rceil C_j$
- **Compartición de recursos**

$$B_i = \max_{\forall k \in lp(i)} (sc_{k,s})$$

$$\forall s \in usa(k) | techo(s) \geq pri(i)$$

$$r_i = C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_j}{T_j} \right\rceil C_j$$
- **Retraso variable en la activación (release jitter)**

$$w_i = C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_j + J_j}{T_j} \right\rceil C_j$$

$$r_i = w_i + J_i$$
- **Tareas aperiódicas**

51

## Análisis básico

\*ARTIS

- Objetivo
- Aportaciones
- Modelo de Entorno
- Arquitectura
- Prioridades
- Prototipo
- Conclusiones
- Extensiones

$$r_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_j}{T_j} \right\rceil C_j$$

$hp(i)$  el conjunto de tareas pertenecientes a niveles estrictamente más prioritarios que  $i$

$r_i$  el tiempo de respuesta de caso peor de la tarea  $i$

$C_i$  el tiempo de cómputo de caso peor de la tarea  $i$

$T_i$  el periodo de la tarea  $i$

la función techo sobre  $x$ , es decir, redondeo de  $x$  al entero superior.

$\lceil x \rceil$

52

## Compartición de recursos

$$B_i = \max_{\forall k \in lp(i)} (sc_{k,s})$$

$$\forall s \in usa(k) | techo(s) \geq pri(i)$$

$$r_i = C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_i}{T_j} \right\rceil C_j$$

*usa(i)*

*techo(s)*

*pri(i)*

*sc<sub>i,s</sub>*

*B<sub>i</sub>*

cjto. de semáforos que usa la tarea *i*

techo de prioridad del semáforo *s*

prioridad de la tarea *i*

longitud de la sección crítica de *i* que envuelve a *s*

duración más larga de bloqueo que puede experimentar *i*

## Release jitter

$$w_i = C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil C_j$$

$$r_i = w_i + J_i$$

- Cuando en el instante que una tarea se activa, ésta no está disponible inmediatamente para ejecución:
    - por el tiempo que tarda el planificador en tomar una decisión,
    - la tarea necesita la llegada de un mensaje pendiente,
    - su periodo no es un múltiplo exacto de la frecuencia del reloj...
- ==> El test puede ser extendido para incluir éste tiempo de retraso variable, conocido como "release jitter time *J<sub>i</sub>*"

# Tareas aperiódicas

\*ARTIS

- Objetivo
- Aportaciones
- Modelo de Entorno
- Arquitectura
- Prioridades
- Prototipo
- Conclusiones
- Extensiones

- Servicio como tareas de 2º plano (background)
  - Servidor de consulta
  - Servidor por Intercambio de Prioridades
  - Servidor Diferido
  - Servidor Esporádico
- ➔ Técnicas de apropiación de holgura
  - Estáticas:
    - SS: Slack Stealer (CMU)
    - SASS: Static Aproximate Slack stealing algorithm (U. York)
  - Dinámicas:
    - DSS: Dynamic slack stealing algorithm (U. York)
    - DASS: Dynamic Aproximate Slack stealing algorithm (U. York)
    - EHDA: Extractor de holgura dinámico aproximado (GTI-IA)

55

# Holgura a nivel

\*ARTIS

- Objetivo
- Aportaciones
- Modelo de Entorno
- Arquitectura
- Prioridades
- Prototipo
- Conclusiones
- Extensiones

La holgura al nivel de prioridad  $i$  es la máxima cantidad de tiempo que se puede dedicar al servicio de tareas aperiódicas a ese nivel de prioridad, sin que falle el plazo de la tarea crítica  $i$ .

■ Tiempo de cómputo  
 ○ Instante de activación  
 ▲ Plazo máximo de ejecución  
 ■ Holgura al nivel de prioridad 3

56

# Cálculo Holgura a nivel 'i'

•ARTIS

- Objetivo
- Aportaciones
- Modelo de Entorno
- Arquitectura
- Prioridades
- Prototipo
- Conclusiones
- Extensiones

holgura = longitud del intervalo - interferencia de las tareas más prioritarias

$$S_i(t) = \left( e - \sum_{j \in hp(i)} I_j(t, e) \right)$$

Interferencia que produce la tarea j en el intervalo:

- cómputo restante de la tarea j en t +
- cómputo de las activaciones completas de la tarea j +
- cómputo (total o parcial) de una posible última activación de la tarea j

$$I_j(t, e) = c_j + f_j(t, e)C_j + \min\left(C_j, \left(e - x_j(t) - f_j(t, e)T_j\right)_0\right)$$

siendo  $f_j(t, e) = \left\lfloor \frac{(e - x_j(t))_0}{T_j} \right\rfloor$  el número de invocaciones completas de la tarea j en el intervalo [t, t+e]

○ Instante de activación

△ Plazo máximo de ejecución

Tempo de cómputo

# Extractor de Holgura Dinámico Aproximado

•ARTIS

- Objetivo
- Aportaciones
- Modelo de Entorno
- Arquitectura
- Prioridades
- Prototipo
- Conclusiones
- Extensiones

Ejecución de la tarea 1:  $I_1: 0 + 3 * 1 + 0 = 3$

Ejecución de la tarea 2:  $I_2: 0 + 2 * 2 + 0 = 4$

Ejecución de la tarea 3:  $I_3: 1$

e: plazo efectivo

|                             |   |   |    |    |
|-----------------------------|---|---|----|----|
| ○ Tiempo de cómputo         | 1 | 1 | 4  | 3  |
| ○ Instante de activación    | 2 | 2 | 6  | 5  |
| △ Plazo máximo de ejecución | 3 | 2 | 24 | 17 |

# Cálculo de la Holgura

Se deben satisfacer los requerimientos de las tareas de cada nivel de prioridad, luego la holgura será:

$$\min_{\forall j \in lp(k)} S_j^{max}(t)$$

•ARTIS

- Objetivo
- Aportaciones
- Modelo de Entorno
- Arquitectura
- Prioridades
- Prototipo
- Conclusiones
- Extensiones

59

# Modelo de asignación de prioridades

•ARTIS

- Objetivo
- Aportaciones
- Modelo de Entorno
- Arquitectura
- Prioridades
- Prototipo
- Conclusiones
- Extensiones

- Activar el servidor inteligente, al acabar una parte inicial y previa comprobación de que existe holgura
- Prioridad (servidor) = prioridad (tarea) cuya parte inicial se completa
- Se han comparado distintas estrategias según:
  - el método de mantenimiento de la holgura que se realiza en el primer nivel de planificación
  - el número de veces que es invocado el método

60

## Estrategia escogida

- ARTIS
  - Objetivo
  - Aportaciones
  - Modelo de Entorno
  - Arquitectura
  - Prioridades
  - Prototipo
  - Conclusiones
  - Extensiones

- Se activa el servidor con  $C_s = \text{holgura}$ , prioridad= i
- Se actualiza la holgura según el tiempo consumido
 
$$\forall j \in lp(i) : S_j(t) = S_j(t') - (t - t')$$
- Activar el servidor ante cada evento significativo, a nivel de la mayor prioridad activa, con  $C_s = \text{holgura}$ .

61

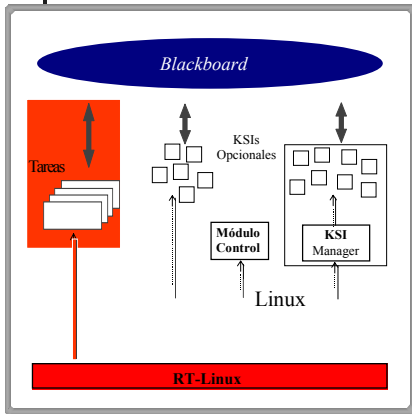
## Ejemplo Estrategia escogida

- ARTIS
  - Objetivo
  - Aportaciones
  - Modelo de Entorno
  - Arquitectura
  - Prioridades
  - Prototipo
  - Conclusiones
  - Extensiones

- Ejecución de la tarea 1
- Ejecución de la tarea 2
- Ejecución de la tarea 3

62

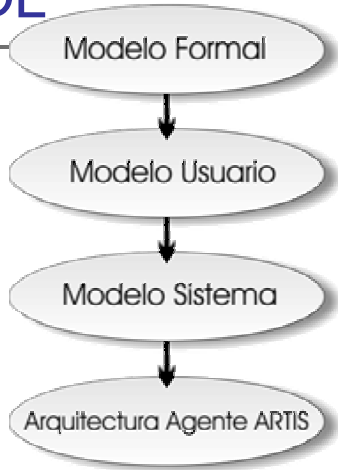
# Prototipo ARTIS



- Aplicación multi-hilo a nivel de usuario.
- Está implementada sobre el sistema operativo RT-Linux
- En el prototipo entran en ejecución dos tipos de componentes :
  - Entidades dependientes de la aplicación (*tareas y KSIs opcionales*).
  - Entidades dependientes de la arquitectura (*RTOS, Módulo de Control*).

# Prototipo RTOS

- El módulo RTOS, núcleo del prototipo, ofrece los siguientes servicios:
  - Activación automática de tareas periódicas.
  - Política de planificación expulsiva basada en prioridades fijas (para las entidades críticas o *tareas*).
  - Soporte para procesamiento opcional, que incluye:
    - » Cálculo del tiempo de holgura.
    - » Llamada al sistema para la ejecución de la parte opcional de las tareas.
    - » Ejecución de KSIs opcionales, de acuerdo con un *segundo nivel de planificación*.
  - Semáforos binarios (*mutexes*) que adoptan el Protocolo del Techo de Prioridad.
  - Colas de mensajes con prioridades, para permitir la comunicación de eventos entre los componentes no críticos del sistema.



The screenshot shows the 'Inside - RobotAgent' application window. The interface includes a menu bar (File, Edit, Entities, Classes, Project), a toolbar, and a main workspace. On the left, a 'Project' tree shows a hierarchy: Entities > InAgents > RobotMovement. The main workspace is divided into several sections:

- Name:** RobotMovement
- Temporal Data:** Period: 200, Deadline: 120, Importance: 1
- Perception / Cognition / Action:** A table with columns for 'MKS' and 'Parameters'. The first row is labeled 'MovementPlanning'.
- Output:** Loading project... Done.



# INSIDE / Lenguaje

**in-agente** → (defagent name  
: **period** int  
: **deadline** int  
: **importance** int  
: **perception** (lista\_llamadas\_MKS's))  
: **cognition** (lista\_llamadas\_MKS's))  
: **action** (lista\_llamadas\_KS)  
: **precedence** (lista\_precedencias))

lista\_precedencias → lista\_prec || λ.  
lista\_prec → (lista\_prec\_MKS's) continuacion\_lista\_prec  
continuacion\_lista\_prec → , lista\_prec || λ.  
lista\_prec\_MKS's → llamada\_MKS llamada\_MKS continuacion\_lista\_prec\_mks  
continuacion\_lista\_prec\_mks → , llamada\_MKS continuacion\_lista\_prec\_mks || λ.  
lista\_llamadas\_MKS's → (obligatoriedad llamada\_MKS) continuacion\_lista\_llamadas || λ.  
continuacion\_lista\_llamadas → , (obligatoriedad llamada\_MKS) continuacion\_lista\_llamadas || λ.  
obligatoriedad → **Mandatory** | **Optional**  
llamada\_MKS → string (lista\_parametros) || string  
lista\_parametros → string continuacion\_lista\_parametros || λ.  
continuacion\_lista\_parametros → , string continuacion\_lista\_parametros || λ.

67



# INSIDE

Inside - RobotAgent

File Edit Entities Classes Project

Project Entities Classes

Entities

- InAgents
  - RobotMovement
  - RobotBatteryLevel
- Mks
  - MovementPlanning
  - SensorReadings
  - CalculateLevel
- Ks
  - firstRead
  - nextMove
  - searchForBetterMovement
  - searchForBestMovement
  - writingEffectors
  - readLevel
  - activeLed

Name: RobotBatteryLevel

Temporal Data

Period: 500  
Deadline: 500  
Importance: 1

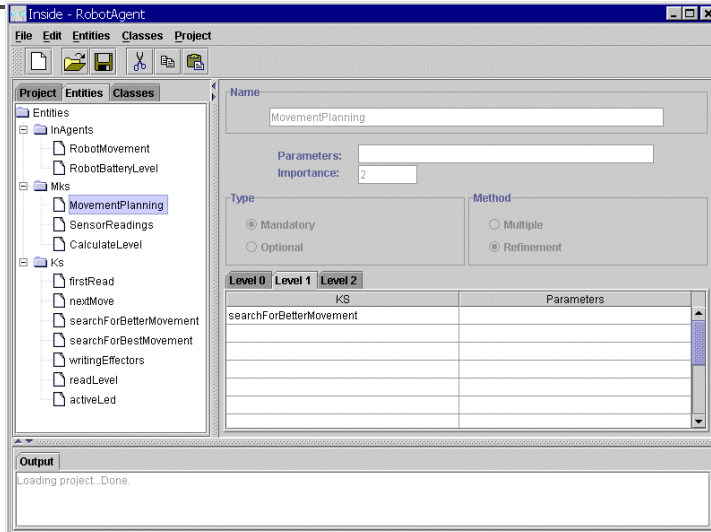
| Perception     | Cognition | Action     |
|----------------|-----------|------------|
| CalculateLevel | MKS       | Parameters |
|                |           |            |
|                |           |            |
|                |           |            |
|                |           |            |
|                |           |            |
|                |           |            |
|                |           |            |
|                |           |            |
|                |           |            |

Output

Loading project... Done.

68

# INSIDE



69

# INSIDE / Lenguaje

**mks** → ( **defmks** *name* ( *lista\_tipo\_parametros* )

*lista\_niveles*

: **importance** int

: **level** **Multiple** | **Refinement**)

*lista\_niveles* → ( *lista\_llamadas\_KS* ) *lista\_niveles* | ( *lista\_llamadas\_KS* )

*lista\_tipo\_parametros* → string *continucion\_lista\_tipo\_parametros* | λ.

*continucion\_lista\_tipo\_parametros* → , string *continucion\_lista\_tipo\_parametros* | λ.

70

# INSIDE

The screenshot shows the 'Inside - RobotAgent' IDE. The left pane displays a project tree with the following structure:

- Entities
  - InAgents
    - RobotMovement
    - RobotBatteryLevel
  - Mks
    - MovementPlanning
    - SensorReadings
    - CalculateLevel
  - Ks
    - firstRead**
    - nextMove
    - searchForBetterMovement
    - searchForBestMovement
    - writingEffectors
    - readLevel
    - activeLed

The main editor displays the implementation of the `firstRead` method:

```
void firstRead() {
    Sensors.North.lighting = getPortValue(3F8h,0);
    Sensors.North.proximity = getPortValue(3F9h,0);
    Sensors.West.lighting = getPortValue(3F8h,1);
    Sensors.West.proximity = getPortValue(3F9h,1);
    Sensors.East.lighting = getPortValue(3F8h,2);
    Sensors.East.proximity = getPortValue(3F9h,2);
    Sensors.South.lighting = getPortValue(3F8h,3);
    Sensors.South.proximity = getPortValue(3F9h,3);
    return;
}
```

The 'Parameters' section shows 'WCET: 40'. The 'Output' pane at the bottom displays 'Loading project... Done.'

71

# INSIDE

The screenshot shows the 'Inside - RobotAgent' IDE with two dialog boxes open over the 'RobotBatteryLevel' class definition.

The 'New Class' dialog box is for 'InfraredSensor' and shows the following configuration:

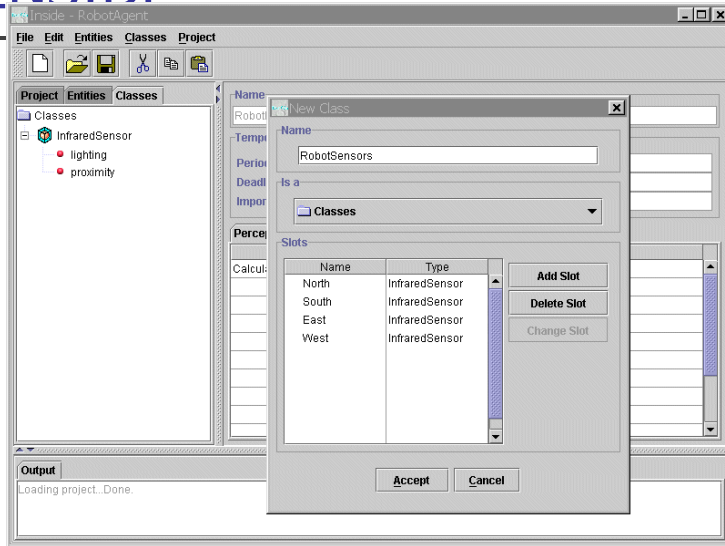
- Name: InfraredSensor
- Is a: Classes
- Slots table:
 

| Name     | Type |
|----------|------|
| lighting | int  |
- Buttons: Add Slot, Delete Slot, Change Slot, Accept, Cancel

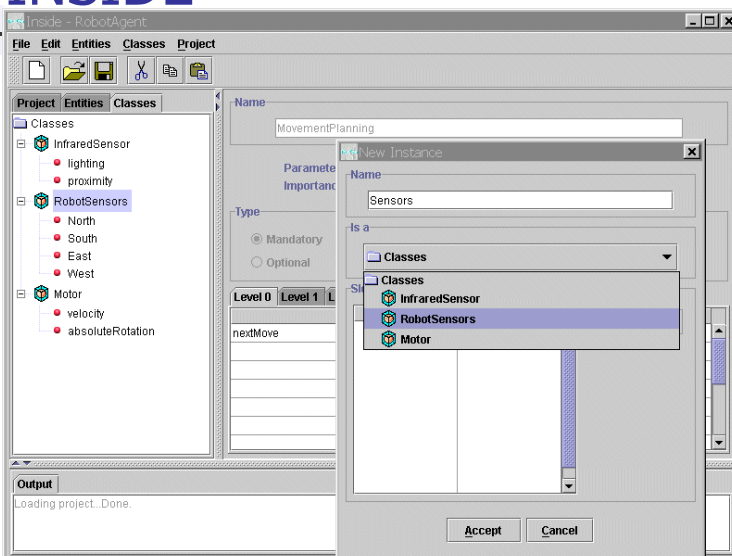
The 'New Slot' dialog box is for the 'proximity' slot and shows the following configuration:

- Name: proximity
- Type: int
- Temporal Properties:
  - Is a temporal Slot
  - Values: 10
  - Predicted Values: (empty)
- Default value: (empty)
- Accessor Read: (empty)
- Accessor Write: (empty)
- Buttons: Accept, Cancel

72

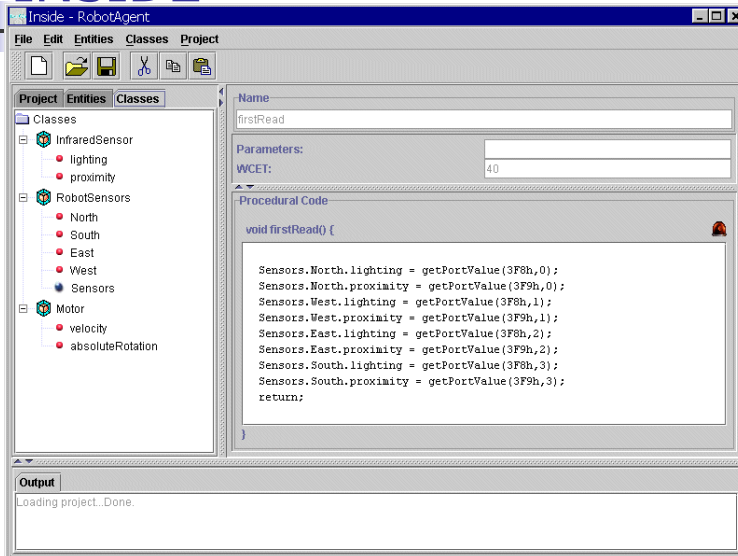


73



74

# INSIDE



75

## Conclusiones

\*ARTIS  
-Objetivo  
-Aportaciones  
-Modelo de Entorno  
-Arquitectura  
-Prioridades  
-Prototipo  
-Conclusiones  
-Extensiones

- En este trabajo se ha presentado un modelo de entidades que permite la incorporación de las técnicas de IA en sistemas de tiempo real y la construcción de un agente en TR, garantizando los tiempos de respuesta de las tareas críticas.
- Para este modelo de entidades se ha diseñado una arquitectura de agente que proporciona la predecibilidad deseada en la construcción de STR.
- Este modelo de entidades junto con la arquitectura propuesta garantiza un mínimo de calidad en la respuesta del agente y en la medida que la carga lo permita, da la posibilidad de mejorar la calidad de dicha respuesta en el tiempo disponible.

76



## Conclusiones (2)

\*ARTIS  
-Objetivo  
-Aportaciones  
-Modelo de Entorno  
-Arquitectura  
-Prioridades  
-Prototipo  
-Conclusiones  
-Extensiones




- Para garantizar los tiempos de respuesta de las tareas críticas del sistema, se utiliza la teoría de planificación por prioridades fijas expulsivas.
- Se ha desarrollado una herramienta de apoyo (INSIDE) para el análisis de la planificabilidad y para el estudio comparativo de distintas estrategias de planificación del 2º nivel.
- Se han identificado las extensiones a los servicios proporcionados por un sistema operativo de tiempo real de base, necesarias para soportar la arquitectura propuesta.

77



## Conclusiones (3)

\*ARTIS  
-Objetivo  
-Aportaciones  
-Modelo de Entorno  
-Arquitectura  
-Prioridades  
-Prototipo  
-Conclusiones  
-Extensiones



- Se ha desarrollado un prototipo del sistema que corre sobre un sistema operativo de amplia difusión (RT-Linux/Linux), implementando el algoritmo de planificación de primer nivel, junto con las extensiones propuestas, a nivel de aplicación
- Se ha realizado un análisis de la planificabilidad de ARTIS para el prototipo construido, donde se ilustra principalmente cómo incluir los componentes de la arquitectura en el análisis.

78

# Extensiones ARTIS

\*ARTIS  
-Objetivo  
-Aportaciones  
-Modelo de Entorno  
-Arquitectura  
-Prioridades  
-Prototipo  
-Conclusiones  
-Extensiones



- Predecibilidad temporal de sistemas basados en reglas
  - Control en tiempo real de agentes guiados por la utilidad
  - Entorno de ejecución
  - Gestión información temporal
  - Planificación en tiempo real
  - Sistema Multiagente / Lenguaje de Comunicación

79

# Predecibilidad temporal de sistemas basados en reglas

\*ARTIS  
-Objetivo  
-Aportaciones  
-Modelo de Entorno  
-Arquitectura  
-Prioridades  
-Prototipo  
-Conclusiones  
-Extensiones



## Análisis del tiempo de respuesta:

- determinar si la ejecución siempre termina en un tiempo acotado
- calcular una cota superior del máximo tiempo de respuesta
- Niveles:
  - acción (off-line, on-line)
  - ejecución de una regla
  - sistema de producción
- Algoritmo de Pattern-matching predecible.

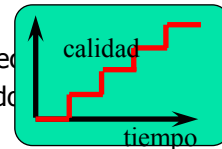
80

# Control en tiempo real de agentes guiados por la utilidad

- ARTIS
  - Objetivo
  - Aportaciones
  - Modelo de Entorno
  - Arquitectura
  - Prioridades
  - Prototipo
  - Conclusiones
  - Extensiones



- Planificación de 2º nivel
  - Incorporar metaconocimiento (o conocimiento de control), y distintas estrategias de planificación de segundo nivel, y sus interacciones con el primer nivel.
  - Métodos para maximizar la calidad de la respuesta (dirigido por los datos) satisfaciendo restricciones de tiempo real.
  - Agente anytime progresivo
  - Balance entre tiempo de deliberación y tiempo de ejecución
- Aumentar incrementalidad, extensión a diferentes periodos



81

# Entorno de Ejecución

- ARTIS
  - Objetivo
  - Aportaciones
  - Modelo de Entorno
  - Arquitectura
  - Prioridades
  - Prototipo
  - Conclusiones
  - Extensiones



- Estudiar la posibilidad de aplicar otras teorías de planificación de STR para garantizar el mínimo nivel de calidad del sistema, y los plazos de respuesta de las tareas críticas.
- Aportar más flexibilidad al sistema identificando y permitiendo distintos modos de funcionamiento.
- Test de garantía on-line para la carga dinámica.
- Adaptación de ARTIS a un sistema distribuido, donde distintos componentes se ejecuten en máquinas diferentes y comunicados a través de una red.
- Desarrollo de núcleo de sistema operativo de tiempo real, que incluya las funcionalidades y comportamientos descritos, y cuyo código se integre en el análisis de la planificabilidad del sistema completo.

82

# Gestión de Información Temporal

\*ARTIS  
-Objetivo  
-Aportaciones  
-Modelo de Entorno  
-Arquitectura  
-Prioridades  
-Prototipo  
-Conclusiones  
-Extensiones



- Necesidad de representación y razonamiento temporal.
- Expresividad adecuada a STR (pasado, presente, futuro, observables, no observables)
- Métodos de propagación y recuperación predecibles.
  - Incrementalidad
  - Soluciones satisfactorias
- Análisis off-line

Tesis Doctoral



ARTIS  
(Interprete SP)

83

# Planificación en Tiempo Real

\*ARTIS  
-Objetivo  
-Aportaciones  
-Modelo de Entorno  
-Arquitectura  
-Prioridades  
-Prototipo  
-Conclusiones  
-Extensiones



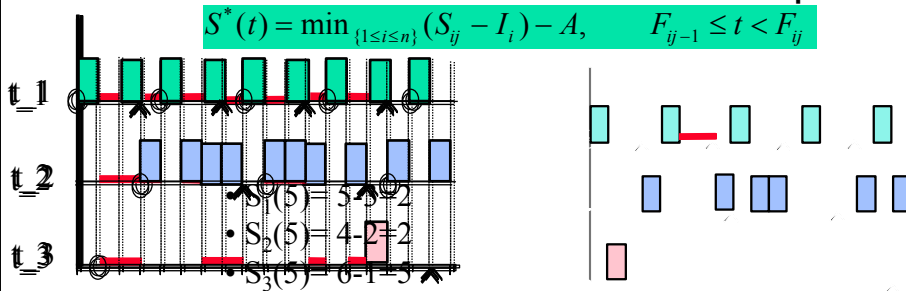
- Agente móvil en un entorno cambiante, no enteramente conocido y con restricciones de tiempo real:
  - conocimiento incompleto
  - objetivos imprecisos y cambiantes
  - planificación de sus acciones en un tiempo acotado
  - balance entre tiempo de deliberación y acción
- Distintos niveles de abstracción
- Monitorización a cada nivel
- Replanificación.

84

## El algoritmo estático de apropiación de holgura: "Slack Stealer" (SS)

| $S_{1j}$ | $F_{1j}$ | $S_{2j}$ | $F_{2j}$ | $S_{3j}$ | $F_{3j}$ |
|----------|----------|----------|----------|----------|----------|
| 2        | 3        | 4        | 8        | 6        | 15       |
| 5        | 7        | 6        | 14       |          |          |
| 8        | 11       |          |          |          |          |
| 11       | 15       |          |          |          |          |

- La holgura disponible en un instante  $t$  viene dada por:



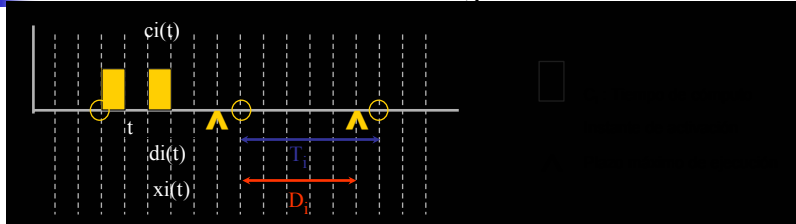
## Comentarios

Aportación: acceso a las tablas por activación

- ¿¿¿ Es óptimo ???
  - minimiza el tiempo de respuesta de las tareas aperiódicas
- bajo coste temporal en tiempo de ejecución,
  - acceso a la tabla y actualización de contadores
- tamaño de la tabla calculada  $n \times H/T_i$
- no puede tratar tareas
  - esporádicas
  - con requisitos de sincronización
  - con posibles retrasos en su activación (requisitos de jitter)

## El algoritmo dinámico de apropiación de holgura: "Dynamic Slack Stealing Algorithm" (DSS)

Conociendo de cada tarea los siguientes datos



- $ci(t)$  tiempo de ejecución restante de la invocación actual de la tarea  $i$ -ésima.
- El objetivo será conocer, para un cierto instante  $t$ , la máxima de tiempo que se puede arrebatar a la tareas pertenecientes al nivel de prioridad  $i$ -ésimo en el intervalo  $[t, t + di(t))$ , sin que ello afecte al cumplimiento de sus plazos máximos de ejecución.

Algoritmos para el cálculo de la holgura

87

## El algoritmo dinámico de apropiación de holgura: "Dynamic Slack Stealing Algorithm" (DSS)

Básicamente el método considera el intervalo  $[t, t + di(t))$  como constituido por una sucesión de periodos pertenecientes a dos tipos:

- Periodo ocupado de nivel  $i$ , que es un intervalo de tiempo continuo en el cual la cola de ejecución contiene una o más tareas con prioridad igual o superior a  $i$ .
- Periodo ocioso de nivel  $i$ , en el cual no hay ejecutándose tareas de prioridad igual o mayor a  $i$ .
- El cálculo de la holgura al nivel de prioridad  $i$  ( $S_i^{\max}(t)$ ) se basa en la identificación de esa sucesión de periodos, y así, la suma de periodos ociosos de nivel  $i$  en el intervalo  $[t, t + di(t))$  nos dará el valor deseado
  - Algoritmo de complejidad  $O(mn^2)$ , siendo  $m$  el número de iteraciones por tarea y  $n$  el número de tareas críticas.

Algoritmos para el cálculo de la holgura

88

$$w_i^{m+1}(t) = S_i(t) + \sum_{\forall j \in hp(i) \cap Y_i} \left( c_j(t) + \left[ \frac{(w_i^m(t) - x_j(t))_0}{T_j} \right] \cdot C_j \right)$$

$$v_i(t, w_i(t)) = \min \left[ \begin{array}{l} (d_i(t) - w_i(t))_0, \\ \min_{\forall j \in hp(i) \cap Y_i} \left( \left[ \frac{(w_i(t) - x_j(t))_0}{T_j} \right] \cdot T_j + x_j(t) - w_i(t) \right) \end{array} \right]$$

89

$$S_i(t) = 0$$

$$w_i^{m+1}(t) = 0$$

**mientras**  $w_i \leq d_i(t)$

$$w_i^m(t) = w_i^{m+1}(t)$$

$$w_i^{m+1}(t) = S_i(t) + \sum_{\forall j \in hp(i) \cap Y_i} \left( c_j(t) + \left[ \frac{(w_i^m(t) - x_j(t))_0}{T_j} \right] \cdot C_j \right)$$

**si**  $w_i^m(t) = w_i^{m+1}(t)$

**entonces**

$$S_i(t) = S_i(t) + v_i(t, w_i(t))$$

$$w_i^{m+1}(t) = w_i^{m+1}(t) + v_i(t, w_i(t)) + e$$

**finsi**

**finmientras**

$$S_i^{\max}(t) = S_i(t)$$

90

## El algoritmo dinámico de apropiación de holgura: "Dynamic Slack Stealing Algorithm" (DSS)

El servicio de las tareas aperiódicas se realizará al nivel de prioridad  $k$ , donde  $k$  la prioridad de la tarea crítica preparada más prioritaria, siempre que exista holgura

$$\min_{\forall j \in lp(k)} S_j^{\max}(t) \geq 0$$

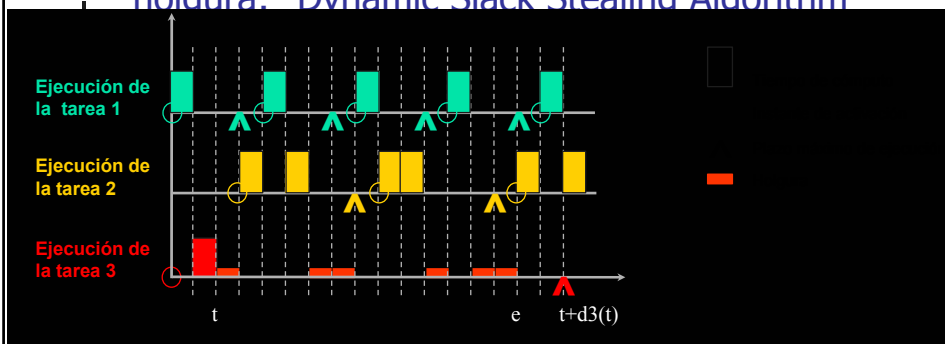
### Extensiones del modelo

- Recuperación del tiempo de cómputo no utilizado
- Compartición de recursos
- Compartición de recursos entre tareas críticas
- Recuperación del tiempo de bloqueo no utilizado
- Compartición de recursos entre tareas críticas y no críticas
- El problema del "Release Jitter"
- Test de aceptación on-line de tareas aperiódicas firmes

Algoritmos para el cálculo de la holgura

91

## El algoritmo dinámico de apropiación de holgura: "Dynamic Slack Stealing Algorithm"



❖ El servicio de las tareas aperiódicas se realizará al nivel de prioridad  $k$ , donde  $k$  la prioridad de la tarea crítica preparada más prioritaria, siempre que exista holgura

$$\min_{\forall j \in lp(k)} S_j^{\max}(t) \geq 0$$

Algoritmos para el cálculo de la holgura

en el número de tareas críticas.

92

- Extensiones del modelo
  - Recuperación del tiempo de cómputo no utilizado
  - Compartición de recursos
  - Recuperación del tiempo de bloqueo no utilizado
  - El problema del "Release Jitter"
  - Test de aceptación on-line de tareas aperiódicas firmes
- Algoritmo óptimo, pero impracticable (alto coste temporal)
- Métodos Aproximados:
  - El algoritmo aproximado estático extractor de holgura SASS
  - El algoritmo aproximado dinámico extractor de holgura DASS

## El algoritmo dinámico de apropiación de holgura: "Dynamic Slack Stealing Algorithm" (DSS)

- Algoritmo óptimo, pero impracticable (alto coste temporal)
- Métodos Aproximados
  - El algoritmo aproximado estático extractor de holgura SASS
    - Calcula una cota inferior de la holgura al nivel de prioridad  $i$  disponible inmediatamente después de la finalización de la tarea  $i$
    - La cota es equivalente al periodo ocioso de nivel  $i$  presente en el intervalo comprendido entre la finalización de la tarea y el plazo de su siguiente invocación
    - Almacena dichos valores en una tabla para cada nivel de prioridad
  - El algoritmo aproximado dinámico extractor de holgura DASS
    - Calcula una cota inferior de la holgura, en tiempo de ejecución
    - La cota equivale a la duración del intervalo de estudio  $d_i(t)$  menos la interferencia que una tarea puede recibir
    - Reducción del grado de pesimismo con el concepto de plazo efectivo

