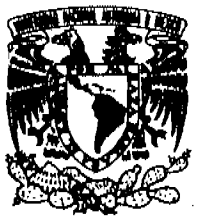


VNIVERSIDAD NACIONAL
AVPNOMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

"DETECCIÓN DE FALLAS Y SU INTEGRACIÓN EN UN SISTEMA AUTOMÁTICO DE OPERACIÓN."

T E S I S

QUE PARA OBTENER EL GRADO DE:

DOCTOR EN CIENCIAS (COMPUTACIÓN)

P R E S E N T A:

M. EN I. DIETER WIMBERGER

DIRECTORA DE TESIS:
DRA. MARÍA CRISTINA VERDE RODARTE

México, D.F. 2008.

To Karl and Irene,
for more than what could possibly fit here.

To Erich,
for the guidance and a better brain OS.

To Alba,
without you I would not have been able to finish this.

To Alejandra,
without you I would never have started this.

To all warriors of the light and the heart.

La perseverancia es favorable.

Agradecimientos y pensamientos

Once again, there are a lot of people to thank for their help, support and encouragement during the time I spend on this work.

First of all Cristina, for her guidance, support and all the lessons learned throughout the time working on the thesis and at the coordination for Automation of the IINGEN in general. Then I would like to thank my tutors, Luis and Fabián for their mentoring efforts at semestral evaluations and other encounters.

I must also acknowledge the general support by the staff at the Postgraduate for Computer Science and Engineering and at the Coordination for Automation. A lot of small things, friendly words and helping hands have contributed to this work directly and indirectly through the atmosphere they created. Staff includes you: Amalia, Lulú and Laura.

Again I owe thanks to the free and open source software community; I have tried my best to retribute to the community in all these last years.

Naturally I would also like to thank all of my family and friends for their support. You all simply were there when needed; thanks that you all exist.

For those that follow, I would like to transmit some thoughts:

> *It's a fact that we learn more when we fail, than when we succeed.*
> (Jonas Ridderstråle and Kjell Nordström)
>
> *At school they only teach us how to accumulate information and they reward memory. Academic teaching does not necesarily stimulate imagination and creativity. Consequently, the ones that obtain the best grades are the ones centered in the past, not in the present or future, or better said, in the wish to triumph.*
> (Paul Arden)
>
> *Try to relax and enjoy the crisis.*
> (Ashleigh Brilliant)

If you make it through the finishing line, remember: the grandness of a human being is determined by the difficulties confronted in everyday life.

> *Before enlightenment, cut wood, carry water. After enlightenment, cut wood, carry water.*
> (Zen proverb)

We are all part of something.

> *Tu alma es todo el universo.*
> (Siddhartha, Hermann Hesse)

Ciudad de México, 15 de mayo de 2008.

Índice general

Resumen

Este trabajo abarca una investigación interdisciplinaria en las áreas de diagnóstico de fallas, arquitectura de sistemas de software y sistemas distribuidos. Para un caso de estudio, un proceso biológico para el tratamiento de aguas residuales en ambientes urbanos e industriales pequeños, se resuelve de manera integral el problema de crear un sistema automático de control y operación (SAOP) con un nivel de supervisión con diagnóstico de fallas (DF), que asegura la confiabilidad en el funcionamiento del proceso.

En la parte del DF, se contribuye una solución genérica a la problemática de la evaluación de la viabilidad de un diagnóstico con un número restringido de sensores. Para el enfoque con base en señales en particular, se aporta una solución para la evaluación y la obtención de un diagnóstico que consiste en la integración de un análisis de sensibilidad, un método de extracción de características y su clasificación mediante clasificadores creados por algoritmos de inteligencia artificial.

En la parte de la realización de un SAOP, la principal contribución es una infraestructura con una arquitectura modular y abierta. Se realizó principalmente como software, permite un manejo de la complejidad en el desarrollo y despliegue como sistema distribuido, y toma en cuenta la necesidad de un SAOP para funcionar en tiempo real, de acuerdo con las propiedades temporales del proceso.

Las soluciones encontradas permiten la aplicación a una familia de problemas con características similares al caso de estudio utilizado: sistemas de pequeña escala que manejan procesos en lotes, en las cuales el costo de inversión es un factor importante para el diseño.

Capítulo 1

Introducción

1.1. General

Hoy en día los procesos automatizados son usados en muchas industrias. No solamente la complejidad de los procesos está aumentando constantemente, sino también la necesidad de eficiencia y confiabilidad en su funcionamiento. Para cumplir con estas necesidades, los sistemas automáticos de control y operación tradicionales se han extendido con mecanismos de supervisión. Estos mecanismos buscan la prevención de un impacto económico o ambiental, mediante la detección y corrección rápida de un comportamiento anormal del sistema.

La Figura 1.1 presenta un esquema general de funciones y niveles de supervisión deseados de un sistema automático de control y operación (SAOP). Sin embargo, hasta ahora solo el nivel de supervisión con monitoreo y protección se ha integrado como estándar en implementaciones de sistemas de control. Este no es el caso del nivel de supervisión con diagnóstico de fallas.

Por un lado, esto se debe a problemas abiertos en el área del diagnóstico de fallas (Dash y Venkatasubramanian, 2000). En la literatura se pueden encontrar propuestas para muchos diferentes métodos de diagnóstico (Isermann, 2005; Venkatasubramanian et al., 2003a,b,c), sin embargo, en la mayoría de las aplicaciones no es una tarea trivial evaluar cuáles fallas pueden ser detectadas y aisladas con un conjunto definido de sensores. Comúnmente esta decisión se toma *a posteriori*, después de elegir un enfoque y método diagnóstico, lo cual es una forma poco eficiente para atacar el problema.

Por otro lado, el aumento en la funcionalidad y el nivel de integración e interacción entre los elementos funcionales de un sistema aumenta de manera proporcional a la complejidad de la tarea de su implementación. En particular, el mecanismo de diagnóstico que busca proporcionar una cierta tolerancia a fallas en el proceso, tiene efectos en el manejo de la operación y requiere la interacción con elementos de otros niveles, el operador (interfaz máquina humano) y la planta (interfaz máquina-máquina).

Heck et al. (2003) y Halang et al. (2006) reconocen el hecho que hoy en día gran parte de la implementación (aproximadamente 60 % al 80 %) de los sistemas automáticos de control y operación de procesos se invierte en el desarrollo de software. Además Halang et al. (2006) hacen notar que las necesidades y requisitos para el desarrollo y la implementación de SAOPs han tenido como consecuencia un acercamiento entre el área de control y el área de tecnologías de información y comunicación. Ellos también describen un conjunto de propiedades que se buscan para el desarrollo y la implementación de un SAOP:

- tolerancia a fallas;

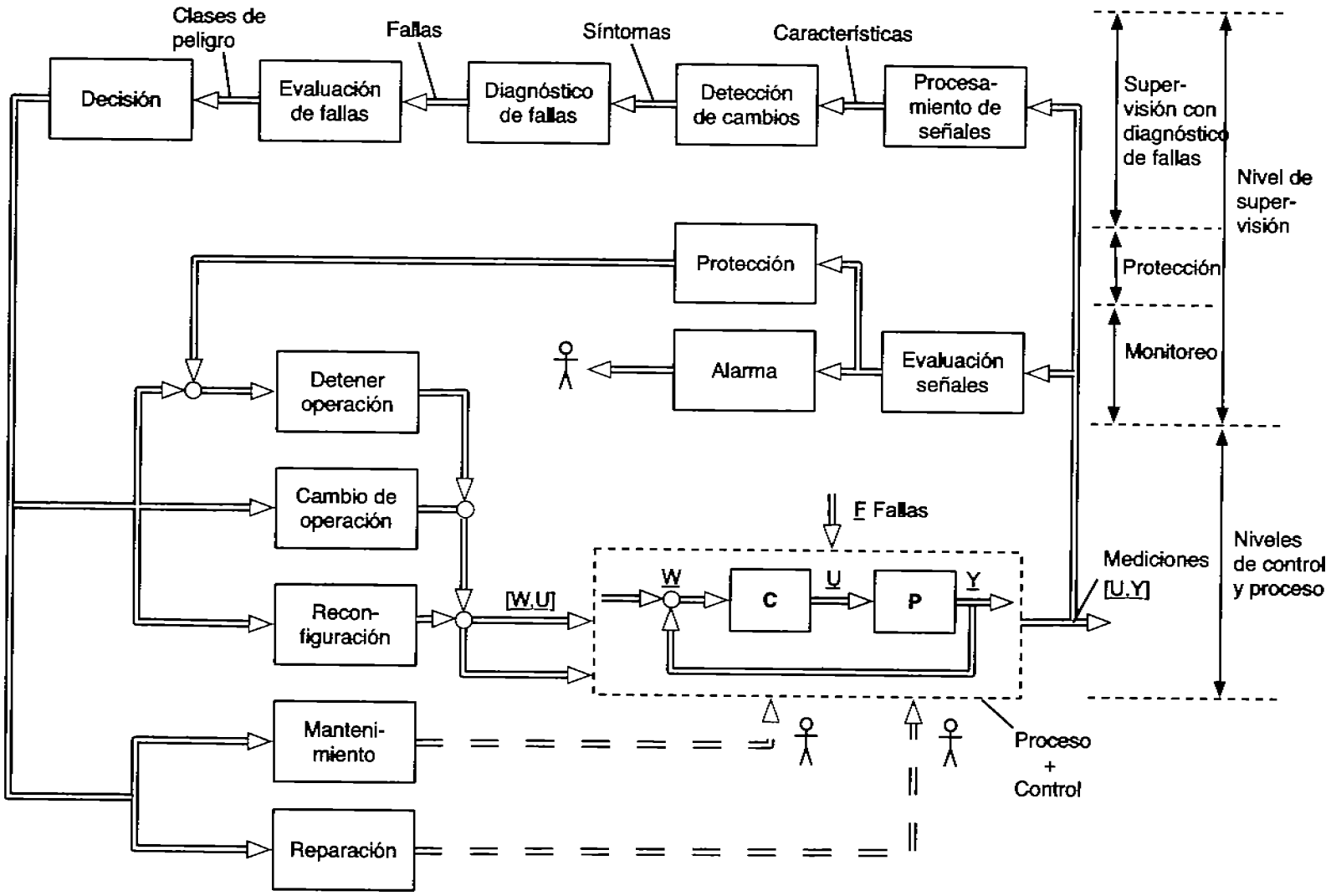


Figura 1.1: Esquema general de funciones y niveles de supervisión de un sistema de control automático (Isermann, 1997)

- distribución;
- modularidad; y
- capacidad para la integración y heterogeneidad.

El énfasis se hace particularmente en soluciones modulares y abiertos, las cuales de acuerdo con Sanz et al. (2000) y Heck et al. (2003), permitan ahorrar tiempo de desarrollo y validación por medio del reuso de módulos para la adaptación del SAOP a:

- diseños nuevos de plantas con diferentes sensores y actuadores; o
- una extensión de una planta durante su ciclo de vida.

Se hace notar que las consideraciones anteriores no solamente afectan a sistemas de gran escala, sino también a sistemas de pequeña escala para un uso:

- industrial, cuando la inversión inicial es un factor importante para el diseño; y
- en investigación y desarrollo, cuando, a medida que un proceso se va validando en el laboratorio, se requieren adaptaciones y extensiones.

El trabajo que aquí se presenta, se basa en un caso de estudio, el cual es un ejemplo donde se requiere la modularidad y bajo costo en inversión para la instalación, operación y mantenimiento del SAOP. Es un proceso biológico para el tratamiento de aguas residuales en ambientes urbanos e industriales pequeños, el cual se describe con detalle en el Capítulo 2. El Capítulo 3, se concentra en resolver el problema de detección y el aislamiento de fallas (DAF) en el proceso del caso de estudio. Sin embargo, al mismo tiempo presenta y aplica metodologías para resolver la problemática de una evaluación *a priori* de la viabilidad de un diagnóstico con métodos analíticos y métodos basados en señales. El diseño y la realización de una infraestructura para SAOP's, que cuenta con las propiedades propuestas por Halang et al. (2006), se encuentra en el Capítulo 4. La infraestructura, que consiste en una arquitectura de software de módulos y mecanismos de comunicación que permiten el transporte de datos en tiempo real suave (Neumann, 2006), se aplica para la integración de un SAOP para el caso de estudio, lo cual se describe con detalle en el Capítulo 5. Finalmente, el Capítulo 6 resume el trabajo con conclusiones y presenta perspectivas de trabajo a futuro.

1.2. Retos

1. Existe una gran variedad de métodos para la detección y el aislamiento de fallas en procesos (Isermann, 2000; Venkatasubramanian et al., 2003a,b,c), los cuales principalmente se distinguen por el uso de diferentes modelos y descripciones de características. Sin embargo, es un problema abierto evaluar la viabilidad de resolver un problema de DAF con un enfoque y método especifico para un caso de estudio.
2. La implementación de un SAOP modular y abierto con un nivel de supervisión que incluye DAF es un problema en sistemas de pequeña escala, donde la inversión inicial es a menudo un factor importante para el diseño. Esto se debe al hecho que productos industriales de compañías o alianzas de compañías, frecuentemente:

   a) no son plataformas abiertas;
   b) requieren una inversión inicial muy grande; y
   c) no están fácilmente disponibles en todas partes del mundo.

3. El diseño de una arquitectura modular, integrada y abierta, que habilite la comunicación entre elementos del sistema:

   a) sin importancia de su colocación física;
   b) garantizando límites en tiempos de retraso; y
   c) permitiendo heterogeneidad.

4. El caso de estudio de este trabajo de investigación en particular requiere:

   a) un sistema automático para una operación autónoma, eficiente y de bajo costo en equipo, instalación y mantenimiento; y
   b) un nivel de supervisión con diagnóstico de fallas para prevenir un impacto económico y ambiental en caso de un comportamiento anormal, que use el mínimo de mediciones y recursos necesarios, para poder realizarse con un bajo costo en plantas de comunidades pequeñas.

   En comparación a procesos de tratamiento de aguas residuales en continuo, hay muy pocos estudios de supervisión con DAF en procesos discontinuos.

1.3. Objetivos

Los problemas abiertos y retos mencionados anteriormente llevaron a formular los objetivos de este trabajo de investigación en:

1. Encontrar un procedimiento adecuado para la evaluación de la viabilidad de un diagnóstico basado en un conjunto definido de sensores, considerando dos enfoques principales y métodos relacionados:

   a) el enfoque basado en modelos analíticos; y
   b) el enfoque basado en modelos con base en datos o señales.

2. Aplicar el procedimiento al caso de estudio, incluyendo

   a) la evaluación de la viabilidad; y
   b) el diseño, la implementación y validación de algoritmos factibles para realizar detección y aislamiento de fallas en el proceso.

3. Desarrollar una infraestructura con una arquitectura modular, integrada y abierta, que permita la realización de un SAOP y proporciona las propiedades requeridas con una solución de bajo costo y fácilmente disponible. Como parte central de la arquitectura se considera la comunicación entre diferentes elementos del sistema:

   a) sin importancia de su colocación física;

b) garantizando límites en tiempos de retraso; y

c) permitiendo heterogeneidad.

4. Verificar las dos partes centrales de este trabajo de investigación de manera integral para el caso de estudio por medio de:

   a) la realización de un SAOP basado en la infraestructura propuesta; y

   b) la realización de un nivel de supervisión con DAF como módulo de este SAOP.

1.4. Contribución

1.4.1. Detección y aislamiento de fallas (DAF)

En la parte de DAF las contribuciones principales consisten en:

1. Un procedimiento para la evaluación de la viabilidad de los dos enfoques determinados como objetivos del trabajo, el cual se puede aplicar a diferentes casos. En particular, incluye un nuevo método para la evaluación de la viabilidad del enfoque con base en señales, el cual:

   a) utiliza la teoría de sensibilidad y se puede aplicar siempre y cuándo exista un modelo analítico; y

   b) en comparación a los métodos cualitativos, los cuales utilizan episodios de tendencias (Charbonnier et al., 2005; Rubio et al., 2004), permite evaluar la viabilidad de un diagnóstico basado en la evolución de señales.

2. El diseño, la implementación y la validación de un algoritmo factible para realizar detección y aislamiento de fallas en el caso de estudio, por medio de la integración de:

   a) un nuevo modelo basado en masas, derivado por medio de una transformación;

   b) un método robusto para la estimación de la tasa de respiración por medio de un observador; y

   c) un método de extracción y clasificación de características de la señal de la tasa de respiración.

1.4.2. Sistema automático de operación

En la parte del SAOP las contribuciones principales consisten en:

1. Una arquitectura basada en soluciones existentes del área de tecnologías de información, la cual:

   a) proporciona modularidad por medio de módulos y abstracciones;

   b) asegura la integración de los módulos sin importar su colocación física; y

   c) considera la problemática de la configuración de un sistema distribuido, que consiste en varios módulos de hardware, en la práctica.

2. La realización de la comunicación por medio de nuevos protocolos que garantizan

   a) un sistema abierto y heterogéneo;

   b) una auto-configuración apta para ambientes de SAOP's; y

   c) un límite en el tiempo de retraso en la comunicación sobre una red de área local realizada con Ethernet, el cual es de bajo costo y fácilmente disponible.

Capítulo 2

Caso de estudio - Reactor para el tratamiento de aguas residuales

2.1. Introducción

Para los habitantes del hemisferio norte el acceso ilimitado a agua limpia comúnmente se considera un hecho, sin embargo, en realidad la mayoría de los actividades humanas presentan un problema con respecto a la disponibilidad de agua en la cantidad y la calidad requerida. Si uno considera adicionalmente, que toda el agua es parte de un ciclo biogeoquímico, aguas contaminadas localmente en algún momento contaminarán el ciclo y eventualmente pueden causar daño a la salud humana y el ambiente. La situación es especialmente crítica con contaminantes como compuestos orgánicos tóxicos, debido a sus efectos en humanos aún en concentraciones bajas. Por lo tanto, aparte del tratamiento físico-químico convencional se requieren procesos biológicos con lodos activos, capaces de eliminar compuestos orgánicos tóxicos y no tóxicos.

Una solución viable al problema de la contaminación de aguas residuales en ambientes urbanos e industriales pequeños, es la descentralización del tratamiento, por medio de pequeñas plantas de tratamiento de agua. Estas plantas tienen que afrontar los siguientes problemas que afectan directamente al proceso:

- una gran variabilidad en el flujo y la composición de las aguas residuales;
- la presencia de sustancias en las aguas residuales que pueden inhibir la actividad de los lodos activos;
- la necesidad de una operación continua, segura, confiable y autónoma;
- la necesidad de un costo bajo en la instalación y operación de la planta.

Estos problemas se pueden resolver con el uso de reactores discontinuos que trabajan por lotes (SBR, por sus siglas en ingles). En comparación con plantas de tratamiento de agua continuas, los SBR (EPA, 1999; Mace y Mata-Alvarez, 2002; Irvine y Ketchum, 1988; Ketchum, 1997) tienen un bajo costo en la construcción e instalación y pueden manejar las variaciones en flujo y composición que ocurren en ambientes urbanos e industriales pequeñas. Sin embargo, para una operación automática segura, confiable y autónoma, se requiere de un buen sistema de operación con supervisión que pueda diagnosticar condiciones anormales. A continuación se describe el proceso que se utilizó como caso de estudio en este trabajo de investigación.

2.2. Modelos

2.2.1. Modelo estructural e instrumentación

Los elementos principales del SBR aeróbico que se estudia, se muestran en la Figura 2.1. Los actuadores y sensores correspondientes se encuentran en la Tabla 2.1.

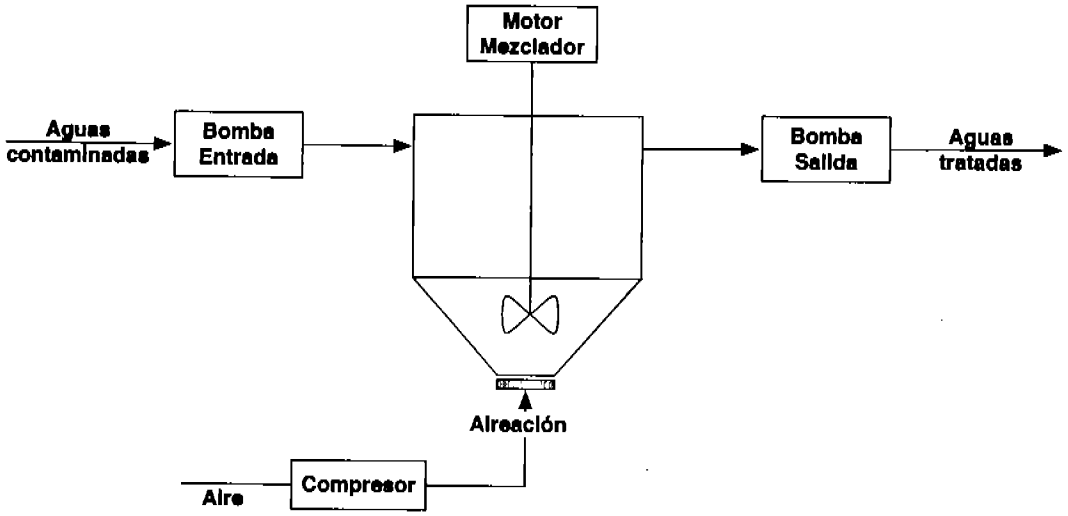


Figura 2.1: Elementos del SBR

Actuadores	*Tipo*	*Sensores*	*Tipo*
Interruptor Bomba de Entrada	D	Estado Bomba de Entrada	D
Interruptor Bomba de Salida	D	Estado de Bomba de Salida	D
Interruptor Mezclador	D	Nivel del Liquido	A
Interruptor Aireación	D	Temperatura	A
Tasa de la Bomba de Entrada	A	Oxígeno Disuelto	A
Tasa de la Bomba de Salida	A	pH	A

Tabla 2.1: Actuadores y sensores (D=digital, A=analógico)

2.2.2. Modelo biológico

El tratamiento de aguas residuales en reactores SBR es un proceso biológico en el cual se utilizan lodos activos. La biología de los lodos activos en el reactor es un aspecto importante del proceso, sin embargo, actualmente no se cuenta con modelos que describan toda la complejidad del sistema biológico correspondiente. Aunque hay trabajos de investigación acerca de métodos de identificación y estudios ecológicos en el contexto del proceso de tratamiento de agua (Wagner et al., 2002; Strous et al., 2002; Purkhold et al., 2000; Daims et al., 2001), estos trabajos todavía no se pueden traducir a modelos genéricos útiles para control y supervisión de un proceso.

En este marco de referencia se propone usar el modelo macroscópico de los lodos activos mostrado en la Figura 2.2. Este modelo se concentra en la capacidad de los lodos activos para transformar algún sustrato del ambiente a otras sustancias, las cuales pueden utilizarse para la reproducción.

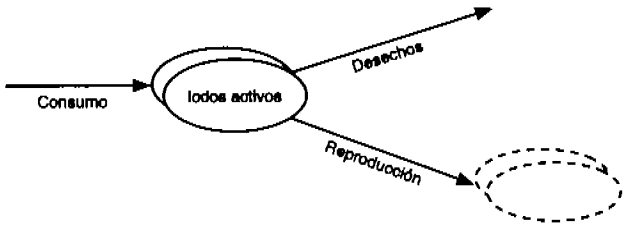


Figura 2.2: Modelo macroscópico de los lodos activos

2.2.3. Modelo bioquímico

El proceso de tratamiento en el caso de estudio es una degradación bioquímica de 4-clorofenol (4-CF). Debido a que es una sustancia orgánica tóxica, existe un límite tolerable de sus emisión al ambiente, el cual fue establecido por la Organización Mundial de Salud (WHO, 1996).

El metabolismo utilizado por los lodos activos para la degradación del 4-CF es básicamente una respiración aeróbica, la cual se puede describir con la siguiente ecuación (Madigan et al., 1999):

$$\text{Sust. orgánica} + O_2 \rightarrow CO_2 + H_2O + \text{Energía (ATP)} \tag{2.1}$$

En este caso el 4-CF es convertido a 4-clorocatecol (los catecoles son intermediarios centrales en la degradación de diferentes compuestos aromáticos), el cual es degradado a compuestos que pueden entrar al ciclo Krebs (dc ácido cítrico), por las dos diferentes vías mostradas en la Figura 2.3.

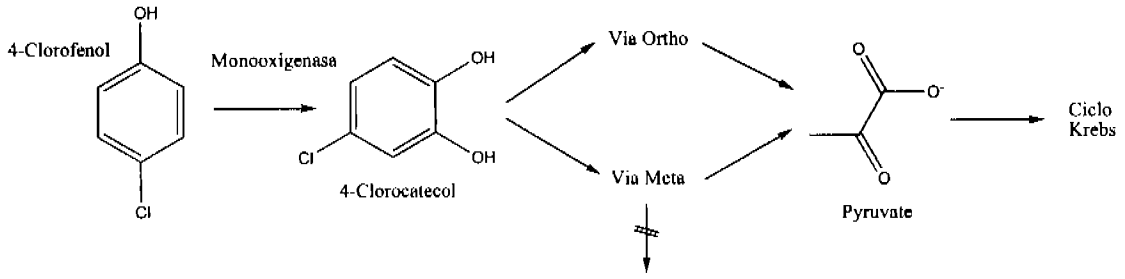


Figura 2.3: Degradación del 4-clorofenol

2.2.4. Modelo de operación

La operación de reactores SBR está basada en el principio "llenar y vaciar" (EPA, 1999) y básicamente consiste en ciclos de tratamiento de un lote de aguas residuales con varias fases, las cuales se describen a continuación y cuya secuencia esta visualizada en la Figura 2.4:

1. *Ayuno:* El reactor no presenta actividad. Los lodos activos se encuentran en suspensión en un volumen mínimo establecido para el reactor.
2. *Re-aireación:* Es la fase inicial del ciclo de operación del reactor SBR; tiene como objetivo la saturación de la suspensión con oxígeno por medio de la aireación.
3. *Llenado y Reacción:* El reactor es llenado con aguas residuales a ser tratadas. El sistema de aireación y el mezclador trabajan continuamente. Para su actividad metabólica los lodos activos utilizan

a) los compuestos orgánicos e inorgánicos en las aguas residuales que entran al reactor; y

b) el oxígeno proporcionado por la aireación.

4. *Decantación*: Se apaga el sistema de aireación y el mezclador el tiempo necesario para sedimentar los lodos activos en la parte del volumen mínimo al fondo del reactor.

5. *Vaciado:* El reactor es vaciado hasta el volumen mínimo definido que contiene los lodos activos en suspensión y el agua tratada saliente del reactor es el producto del proceso.

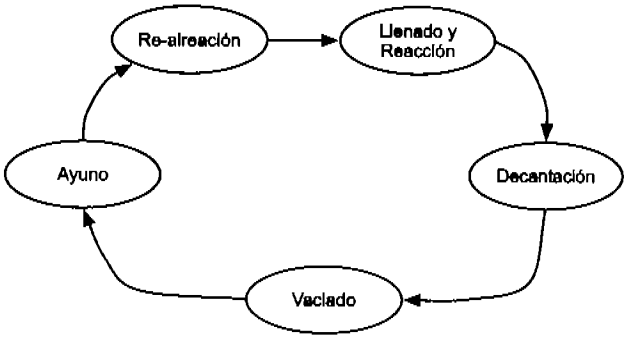


Figura 2.4: Secuencia de los fases de un ciclo de tratamiento en el reactor SBR

2.2.5. Modelo analítico de la fase de llenado y reacción

Los modelos de concentración son comunes en el área de biotecnología (Henze et al., 2000) y usualmente se aplican bajo la suposición de que el volumen es constante, lo cual sólo es el caso en procesos continuos y procesos en lotes idealizados. Bajo la suposición del volumen constante, estos modelos son equivalentes a modelos de balances de masas. Sin embargo, a una escala industrial el llenado de un SBR no es instantáneo y ocurre al mismo tiempo con una parte de la reacción. La suposición del volumen constante ya no es válido y el término de la dilución puede causar problemas en la aplicación del modelo basado en balances de concentraciones.

Para evitar este problema en procesos químicos, Hangos y Cameron (2001) proponen el uso exclusivo de modelos basados en cantidades extensivas conservadas. Por ello, el modelo de Betancur et al. (2004) utilizado como base en este trabajo, se transformó a un modelo basado en balances de masas descrito por las ecuaciones (2.2) y (2.3).

$$\begin{aligned} \frac{dm_x}{dt} &= \mu_m\, m_x \\ \frac{dm_s}{dt} &= -k_1 \mu_m\, m_x + q_{in} c_{s,in} \\ \frac{dm_o}{dt} &= r_m + q_{in} c_{o,in} + K_{la}(m_{o,sat} - m_o) \\ \frac{dV}{dt} &= q_{in} \end{aligned} \tag{2.2}$$

donde $m_x, m_s, m_o, V, \mu_m, k_1, q_{in}, c_{s,in}, r_m, c_{o,in}, K_{la}, m_{o,sat}$ son la masa de los lodos activos, la masa del sustrato y la masa de oxígeno disuelto, el volumen de la fase líquida, la tasa de crecimiento

en masa, el coeficiente de conversión de sustrato a masa de lodos activos, el flujo de entrada, la concentración en el flujo de entrada, la tasa de respiración de oxígeno en masa, la concentración de oxígeno en el flujo de entrada, el coeficiente de la transferencia de oxígeno y la masa de oxígeno de saturación, respectivamente.

La tasa de crecimiento es descrito por un modelo de Andrews (1968):

$$\mu_m = \frac{\mu_0 m_s}{K_s + m_s + \frac{m_s^2}{K_i}} \tag{2.3}$$

donde μ_0, K_s, K_i son la tasa de crecimiento especifico en masa, el coeficiente de media saturación y el coeficiente de inhibición, respectivamente.

La tasa de respiración es una función de μ_m y m_x que consta de una parte metabólica y una endógena:

$$r_m = -k_2 \mu_m \, m_x - b m_x \tag{2.4}$$

donde k_2 y b son el coeficiente de conversión de oxígeno a masa de lodos activos y el coeficiente de la respiración endógena, respectivamente.

Datos de experimentos de identificación, obtenidos en el laboratorio del Instituto de Ingeniería de la UNAM, han sido utilizado para obtener los valores nominales del los parámetros y coeficientes del modelo de concentraciones (Betancur et al., 2005). Estos valores, presentados en la Tabla 2.2, se pueden utilizar en Ec. (2.2) y (2.3), bajo la suposición que el volumen en el reactor está disponible mediante una medición en línea. El comportamiento nominal de los cuatro estados del sistema m_x, m_s, m_o y V durante el tiempo de llenado y reacción se presenta en la Figura 2.5.

T_{react}	=	$1{,}8\,[h]$	$c_{o,sat}$	=	$7{,}033\,[mg/l]$
$c_x(0)$	=	$4734\,[mg/l]$	K_{la}	=	$16{,}8\,[h^{-1}]$
$c_s(0)$	=	$0\,[mg/l]$	K_i	=	$3{,}753\,[mg/l]$
$c_o(0)$	=	$7{,}033\,[mg/l]$	K_s	=	$60\,[mg/l]$
$V_r(0)$	=	$3\,[l]$	μ_0	=	$0{,}1916\,[h^{-1}]$
q_{in}	=	$14{,}8\,[mg/l]$	k_1	=	$3{,}7$
$c_{s,in}$	=	$168\,[mg/l]$	k_2	=	$1{,}0363$
$c_{o,in}$	=	$0\,[mg/l]$	b	=	$0{,}0059\,[h^{-1}]$

Tabla 2.2: Parámetros y condiciones iniciales nominales

Dado el requerimiento de bajo costo de inversión en la operación del reactor en el caso de estudio, se supone que solamente se pueden medir en línea las siguientes variables:

1. el volumen del reactor;
2. la concentración del oxígeno disuelto; y
3. la temperatura.

Adicionalmente se supone que la tasa de flujo de entrada q_{in} y la concentración de oxígeno en el flujo de entrada $c_{o,in}$ son conocidas y constantes durante la fase de llenado y reacción, lo cual es una suposición razonable en el caso de estudio.

Finalmente, se hace notar el hecho, que en este caso están disponibles pocos datos históricos de operación y hay ausencia de datos de operación en casos con presencia de fallas.

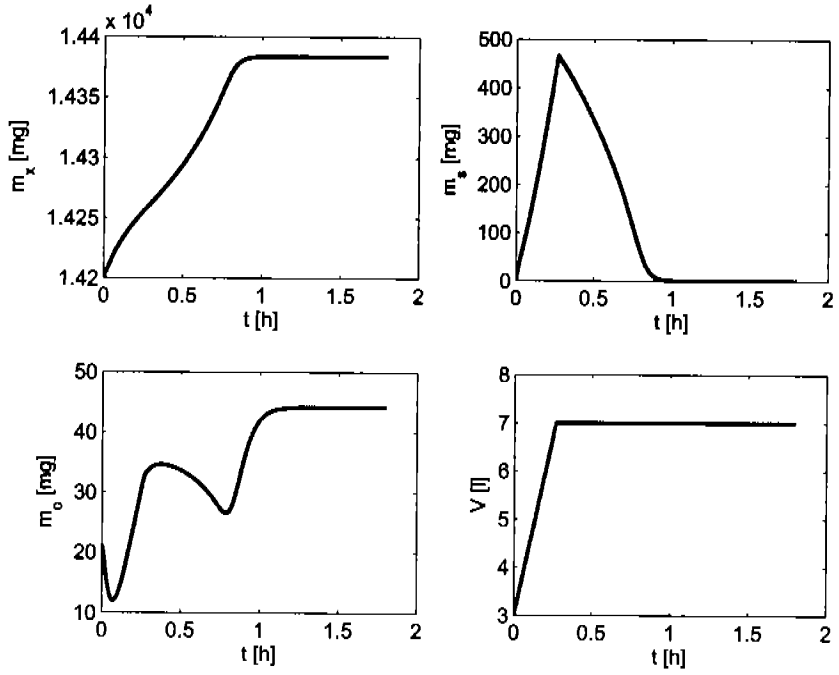


Figura 2.5: Evolución de los estados bajo condiciones nominales en una simulación

2.3. Resumen de características generales del caso de estudio

El caso de estudio presentado se puede enmarcar en un proceso con las siguientes características generales:

- es un proceso secuencial y en lotes;
- la estructura de un modelo analítico es conocida;
- existen incertidumbres con intervalos conocidos en condiciones iniciales y parámetros;
- las dinámicas del proceso son relativamente lentas, se pueden describir con muestreos en un orden de segundos;
- el número de sensores y actuadores es relativamente pequeño (alrededor de 20) y consecuentemente están disponibles pocas mediciones en línea para realizar un monitoreo y/o diagnóstico;
- están disponibles pocos datos históricos de operación y hay ausencia de datos en casos de fallas; y
- se busca lograr un bajo costo de inversión para construcción, operación y mantenimiento con un sistema automático de operación.

Capítulo 3

Diagnóstico de fallas

3.1. Introducción

La operación de un proceso es confiable cuando esta se puede mantener en presencia de fallas, las cuales están definidas como una desviación no permitida de al menos una propiedad característica o de un parámetro del sistema de su condición nominal (Isermann y Ballé, 1997). Para lograr la confiabilidad, el proceso requiere de un sistema de supervisión con diagnóstico de fallas, el cual puede detectar la presencia de una falla, identificar su tipo y tomar la decisión de notificar a un operador o reconfigurar el sistema automáticamente (Figura 1.1).

El esquema clásico de detección de fallas (Isermann y Ballé, 1997) busca detectar y aislar fallas que ocurren en sensores, actuadores y el proceso al mismo tiempo. No obstante, con los avances tecnológicos muchos sensores y actuadores ya incluyen un tipo de "inteligencia" que proporciona un sistema de auto-diagnóstico al nivel del equipo y determina si su propio comportamiento está dentro del comportamiento nominal. Este hecho tiene como consecuencia un esquema distribuido de diagnóstico como el que se presenta en la Figura 3.1.

En este trabajo se supone que la determinación del estado de operación de los sensores y actuadores se realiza de manera independiente, siguiendo el esquema de la Figura 3.1. Por lo tanto, se concentrará en resolver el problema de la detección y el aislamiento de fallas (DAF) en el proceso del caso de estudio. En particular, dado que el proceso tiene como objetivo la degradación de una sustancia orgánica tóxica peligrosa para el ambiente, el humano y en ciertos casos también para los lodos activos, se busca un método para detectar y aislar fallas en la fase de llenado y reacción del proceso como son:

- cambios en las características extrínsecas de los lodos activos, las cuales tienen como consecuencia en el comportamiento bioquímico;
- la degradación incompleta del 4-CF, la cual pueda tener como consecuencia la contaminación del ambiente, ya que permanece en concentraciones demasiado altas en el agua tratada que sale del reactor.

En la literatura se pueden encontrar diferentes métodos de DAF (Isermann, 2005; Venkatasubramanian et al., 2003a,b,c). Sin embargo, no es una tarea trivial evaluar cuales fallas pueden ser detectadas y aisladas con un conjunto definido de sensores en la mayoría de las aplicaciones. Comúnmente esta decisión se toma *a posteriori*, después de elegir un enfoque y método de diagnóstico, lo cual es una

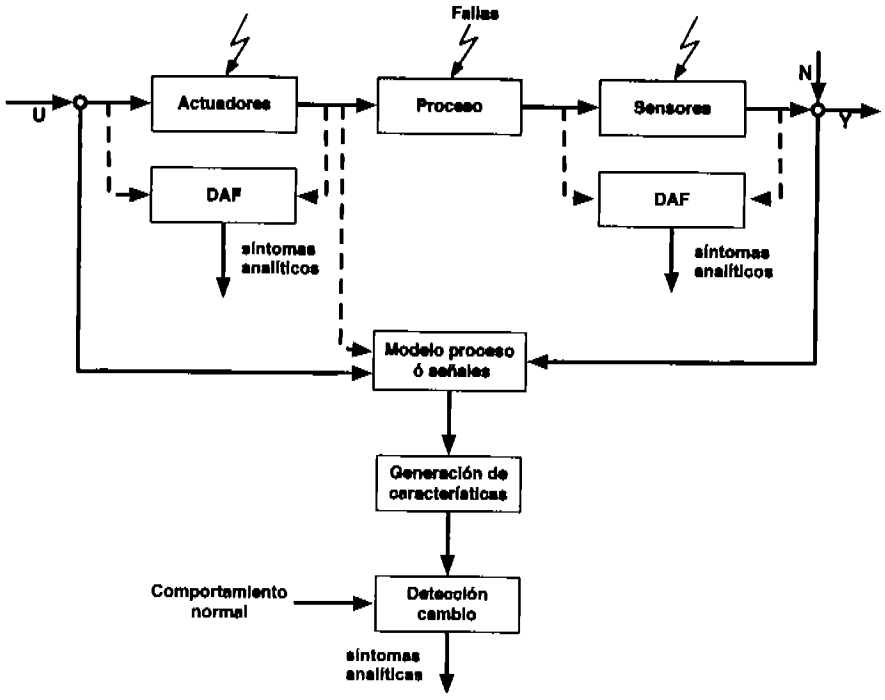


Figura 3.1: Esquema distribuido de DAF, el cual presenta una derivación del esquema clásico de detección de fallas presentado por Isermann y Ballé (1997).

forma poco eficiente para atacar el problema. En este trabajo la problemática de una evaluación *a priori* de la viabilidad de un diagnóstico es de gran importancia.

Para evaluar la viabilidad de un diagnóstico en un problema específico con métodos analíticos de diagnóstico, se puede aplicar el análisis estructural (Blanke et al., 2003). El caso de DAF con base en modelos basados en señales (Isermann, 2005) es una problemática diferente. Para procesos secuenciales en particular, el diagnóstico basado en PCA (Kourti, 2003) o tendencias transformadas a elementos sintácticos cualitativos (Rubio et al., 2004) solo permite una evaluación *a posteriori*, usando posibles algoritmos de diagnóstico ya implementados.

Adicionalmente, el punto de partida para métodos basados en señales son datos históricos que cubren casos de comportamientos normales y anormales del proceso (Isermann, 2005; Kourti, 2003). Debido al hecho que estos datos no están disponibles en el caso de estudio, se propone un método basado en la estructura de un modelo analítico conocido y en un análisis derivado de la teoría de sensibilidad (Frank, 1978). Se muestra que la interpretación de los resultados de este análisis permite la evaluación *a priori* y, por lo tanto, que es un método adecuado para resolver éste problema en casos en los cuales se cuenta con la estructura de un modelo analítico.

La Figura 3.2 presenta los pasos de la metodología propuesta para llegar a una solución del problema de diagnóstico en el caso de estudio:

1. La evaluación *a priori* de la viabilidad del diagnóstico usando

   a) el enfoque basado en métodos analíticos y

   b) el enfoque basado en datos o señales.

2. La formulación del problema de diagnóstico, basado en los resultados de la evaluación descrita anteriormente para:

   a) la decisión del enfoque que se va a utilizar;

   b) las fallas e incertidumbres a tratar.

3. La selección de un método de DAF, para la cual se toma en cuenta la información de los resultados de la evaluación de viabilidad.

4. La validación del método de DAF propuesto.

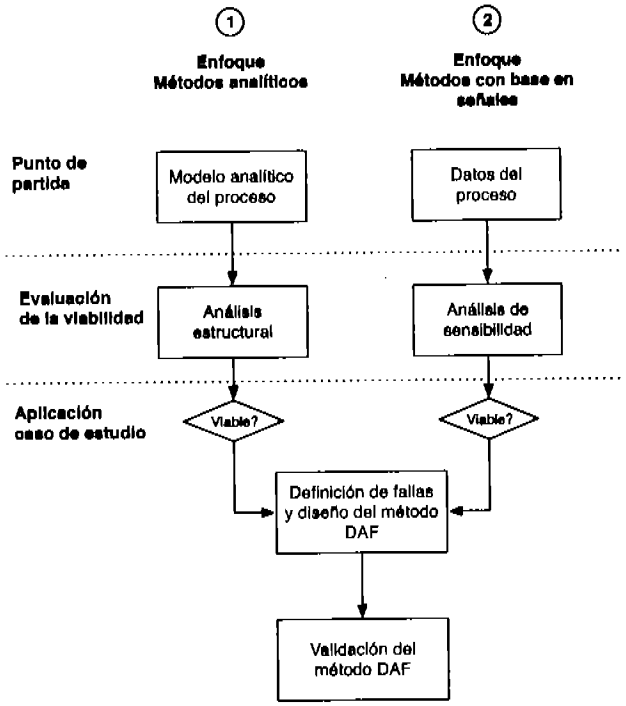


Figura 3.2: Pasos de la metodología para el DAF

3.2. Evaluación de la viabilidad de un diagnóstico

3.2.1. Métodos analíticos

3.2.1.1. Análisis estructural

La disponibilidad de un modelo analítico del proceso no garantiza que sea posible diseñar un algoritmo de DAF para diagnosticar ciertos tipos de fallas. Blanke et al. (2003) presentan el análisis estructural como una herramienta para la evaluación de posibilidades de diagnóstico, asumiendo conocida la estructura del proceso. El primer paso del análisis es la extracción de un modelo estructural, el cual refleja las relaciones entre variables a través de las ecuaciones del modelo analítico. La ventaja de disponer de la estructura del proceso es, que el análisis es aplicable a casi todos los modelos analíticos con la misma estructura.

En particular, Blanke et al. (2003) proponen representar la estructura del modelo por medio de una gráfica bipartida

$$G = (\mathcal{V} \cup \mathcal{C}, \Gamma) \tag{3.1}$$

donde $\mathcal{V} = \{v_1, v_2, ..., v_k\}$ es el conjunto de vértices que corresponde a las variables que aparecen en las restricciones, $\mathcal{C} = \{c_1, c_2, ..., c_m\}$ es el conjunto de vértices que corresponde a las restricciones y

$$\Gamma = \{(c_j, v_j) \mid v_j \text{ es parte de } c_i,\ i = \{1, 2, .., m\},\ j = \{1, 2, ..., k\}\} \tag{3.2}$$

es el conjunto de aristas. La gráfica bipartida se puede representar con una matriz M cuyas filas corresponden a $\mathcal{C}$ y cuyas columnas corresponden a $\mathcal{V}$, tal que, los elementos de M toman los siguientes valores:

$$m_{i,j} = \begin{cases} 1 & \text{si } (c_i, v_j) \in \Gamma \\ 0 & \text{si } (c_i, v_j) \notin \Gamma \end{cases} \tag{3.3}$$

La matriz M se denomina *matriz de incidencia de la gráfica estructural.* Su descomposición canónica es el punto de partida del análisis. La descomposición divide el modelo estructural representado por M en tres partes: uno estructuralmente sobre-determinado, denominado M^+, uno estructuralmente determinado M^0 y uno estructuralmente sub-determinado M^- (Blanke et al., 2003). Para el análisis relacionado al diagnóstico de fallas, sólo se considera la parte sobre-determinada, debido al hecho que es la única redundancia entre las variables, la cual es la base de todo sistema diagnóstico.

3.2.1.2. Análisis estructural del modelo de llenado y reacción

Siguiendo la propuesta de Blanke et al. (2003) las derivadas de las variables de estado en un sistema dinámico deben ser incluidas como restricciones y consideradas como variables. Por lo tanto, para el modelo del SBR (Ec. 2.2 y 2.3), se puede obtener el conjunto $\mathcal{C}$ de restricciones del sistema:

$$\begin{array}{lllll}
c_1: & \dot{m}_x & = & \mu_m m_x \\
c_2: & \dot{m}_s & = & -k_1 \mu_m m_x + q_{in} c_{s,in} \\
c_3: & \dot{m}_o & = & r_m + K_{la}(m_{o,sat} - m_o) + q_{in} c_{o,in} \\
c_4: & \dot{V}_r & = & q_{in} \\
c_5: & \mu_m & = & \frac{\mu_Q m_s}{K_s + m_s + \frac{m_s^2}{K_i}} \\
c_6: & m_{o,sat} & = & c_{o,sat} V_r \\
c_7: & c_{o,sat} & = & f(T) \\
c_8: & \dot{m}_x & = & \frac{dm_x}{dt} \\
c_9: & \dot{m}_s & = & \frac{dm_s}{dt} \\
c_{10}: & \dot{m}_o & = & \frac{dm_o}{dt} \\
c_{11}: & \dot{V}_r & = & \frac{dV_r}{dt} \\
c_{12}: & m_o & = & c_o V_r \\
c_{13}: & r_m & = & -k_2 \mu_m m_x - b m_x
\end{array} \tag{3.4}$$

El conjunto de variables está definido por

$$\mathcal{V} = \{q_{in}, T, c_o, V_r, c_{s,in}, c_{o,in}, c_{o,sat}, m_x, m_s, m_o, \mu_m, m_{o,sat}, \dot{m}_x, \dot{m}_s, \dot{m}_o, \dot{V}_r\} \tag{3.5}$$

el cual contiene el conjunto de variables desconocidas

$$\mathcal{X} = \{m_x, m_s, \mu_m, \dot{m}_x, \dot{m}_s, \dot{m}_o\} \tag{3.6}$$

y de las variables conocidas $\mathcal{K} = \mathcal{V} \backslash \mathcal{X}$.
La estructura en M^+, y eliminando las variables conocidas, se reduce a:

	m_x	m_s	μ_m	$\dot{m}_x$	$\dot{m}_s$	$\dot{m}_o$	r_m
c_1	1	0	1	1	0	0	0
c_2	1	0	1	0	1	0	0
c_5	0	1	1	0	0	0	0
c_8	1	0	0	1	0	0	0
c_9	0	1	0	0	1	0	0
c_{10}	0	0	0	0	0	1	0
c_3	0	0	0	0	0	1	1
c_{13}	1	0	1	0	0	0	1

(3.7)

La descomposición canónica de (3.7) permite generar un emparejamiento máximo del sistema, el cual se puede traducir a la gráfica bipartida presentada en la Figura 3.3.

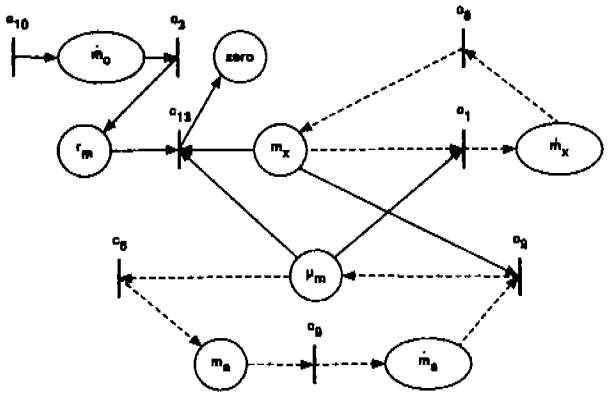


Figura 3.3: Gráfica bipartida de un emparejamiento máximo obtenida por descomposición canónica (aristas discontinuas son lazos con diferenciales; la monitoreabilidad de r_m se muestra en la parte azul de la gráfica).

La presencia de lazos diferenciales con $\dot{m}_s$ y $\dot{m}_x$ indica que el cálculo de m_s y m_x es sensible con respecto a las condiciones iniciales $m_s(0)$ y $m_x(0)$. En el caso de estudio esto resulta crítico, ya que el valor de $m_x(0)$ es incierto.

3.2.1.3. Análisis estructural de un observador asintótico

Otra opción para diagnosticar fallas en el proceso consiste en integrar sensores por software u observadores. En el caso de estudio, basado en el modelo Ec. (2.2), se puede derivar un observador asintótico usando una transformación particular, similar a la propuesta de Bastin y Dochain (1990). Este observador permite la estimación de m_x y m_s sin conocimiento explicito de la tasa de crecimiento μ_m. La transformación se basa en dos nuevos estados z_1 y z_2 :

$$z_1 = m_o + k_2 m_x \tag{3.8}$$

$$z_2 = m_o + \frac{k_2}{k_1} m_s \tag{3.9}$$

Los valores de m_x y m_s se pueden extraer y sustituir en el modelo Ec. (2.2), lo cual da como resultado las ecuaciones del observador:

$$\frac{dz_1}{dt} = -b\frac{z_1 - m_o}{k_2} + K_{la}(m_{o,sat} - m_o) + q_{in}c_{o,in} \tag{3.10}$$

$$\frac{dz_2}{dt} = -b\frac{z_1 - m_o}{k_2} + K_{la}(m_{o,sat} - m_o) + q_{in}c_{o,in} - q_{in}c_{s,in}\frac{k_2}{k_1} \tag{3.11}$$

La viabilidad del diagnóstico basado en el observador, se puede evaluar también aplicando el análisis estructural. El conjunto de restricciones $\mathcal{C} = \{c_1, ..., c_6\}$ del sistema en consideración se muestra en Ec. (3.12). El conjunto asociado de variables es $\mathcal{V} = \{z_1, z_2, m_x, m_s, \dot{z_1}, \dot{z_2}\}$.

$$\begin{array}{lrcl} c_1: & z_1 & = & m_o + k_2 m_x \\ c_2: & z_2 & = & m_o + \frac{k_2}{k_1} m_s \\ c_3: & \dot{z_1} & = & -b\frac{z_1 - m_o}{k_2} + K_{la}(m_{o,sat} - m_o) + q_{in}c_{o,in} \\ c_4: & \dot{z_2} & = & -b\frac{z_1 - m_o}{k_2} + K_{la}(m_{o,sat} - m_o) + q_{in}c_{o,in} - (q_{in}c_{s,in})\frac{k_2}{k_1} \\ c_5: & \dot{z_1} & = & \frac{dz_1}{dt} \\ c_6: & \dot{z_2} & = & \frac{dz_2}{dt} \end{array} \tag{3.12}$$

La descomposición canónica de Ec. (3.12) arroja que el sistema consiste solamente en una parte estructuralmente determinada M^0, lo cual significa que no hay redundancia en la estructura y por lo tanto el sistema (3.12) no puede ser utilizado para resolver un problema de DAF.

Los resultados del análisis anterior indican que con las mediciones y el modelo analítico disponibles en el proceso del caso de estudio, no es posible diseñar un algoritmo robusto de DAF con respecto a incertidumbres en m_x y m_s basado en métodos analíticos de diagnóstico.

3.2.2. Métodos con base en señales

Una alternativa diferente al enfoque analítico de DAF es atacar el problema con un método que se basa en un modelo basado en una o múltiples señales. Mientras que hay muchos métodos basados en la comparación de patrones en señales con y sin presencia de fallas (Isermann, 2005; Venkatasubramanian et al., 2003b,c), la evaluación *a priori* de la viabilidad de detección y aislamiento de fallas particulares es un problema abierto.

3.2.2.1. Teoría de sensibilidad

Considerando que en el caso de estudio está disponible un modelo analítico, se propone explotarlo para evaluar *a priori* la viabilidad de DAF usando la teoría de sensibilidad (Frank, 1978). Esta teoría proporciona una base para evaluar el efecto de desviaciones pequeñas en parámetros y condiciones iniciales en el comportamiento de un sistema y, como consecuencia, de sus señales.

En particular, la función de la sensibilidad de una variable, con respecto a la desviación de un parámetro y con respecto al tiempo está definida por

$$\sigma(t, \alpha_0) \triangleq \frac{\partial y(t, \alpha)}{\partial \alpha} \mid_{\alpha=\alpha_0} \tag{3.13}$$

donde y es la variable, α es el parámetro que se desvía y α_0 es el valor nominal del parámetro. Análogamente se puede obtener la función de la sensibilidad de una variable con respecto a la desviación de una condición inicial y con respecto al tiempo.

Para obtener la evolución de $\sigma(t, \alpha_0)$, Frank (1978) propone dos métodos:

- un método analítico; y
- un método estructural que consiste en la medición de sensibilidades por medio de simulaciones numéricas.

Para los resultados presentados en este trabajo se utilizó una herramienta denominada ODESSA (Leis y Kramer, 1988), la cual está disponible en el paquete de software denominado OCTAVE (Eaton, 2002) y permite al mismo tiempo obtener las soluciones para un sistema de ecuaciones diferenciales ordinarias y las funciones de sensibilidad asociadas por medio de simulaciones numéricas.

Para hacer comparables las diferentes funciones de sensibilidad de una variable con respecto a diferentes parámetros y de condiciones iniciales, se propone normalizar con base en la variable. Suponiendo $y(t, \alpha_0) \neq 0$ esta normalización se describe con

$$\overline{\sigma} = \frac{\sigma(t, \alpha_0)}{y(t, \alpha_0)} \tag{3.14}$$

Para la evaluación de la viabilidad de un diagnóstico basado en una variable o señal en particular, se propone la siguiente metodología:

1. Definición de un conjunto de parámetros y condiciones iniciales en las cuales se consideran desviaciones de los valores nominales;
2. Obtener las funciones de sensibilidad normalizadas de la variable con respecto a la desviación de los parámetros y condiciones iniciales del conjunto anterior;
3. Comparar cualitativamente la evolución de las funciones de sensibilidad normalizadas para encontrar puntos y regiones donde existen diferencias significativas en tiempo y amplitud de efectos causados por la desviación;
4. Utilizar las observaciones de la comparación anterior para definir dos subconjuntos del conjunto definido en el primer paso:
   a) un conjunto de parámetros y condiciones iniciales de las cuales toda desviación del valor nominal se define como incertidumbre;
   b) un conjunto de parámetros y condiciones iniciales de las cuales la desviación del valor nominal a partir de un cierto tamaño se define como falla.

Se hace notar, que las observaciones del paso tres también arrojan como resultado la identificación de las características de la señal que van a permitir la detección y el aislamiento de fallas definidas en 4 a). Este hecho se va a utilizar en la propuesta de la solución al problema diagnóstico.

3.2.2.2. Estimación de la tasa de respiración

El consumo de oxígeno de los lodos activos para realizar su actividad metabólica (Sección 2.2.3) es la tasa de respiración r_m. La tasa de respiración es reconocida como un indicador de la actividad de los lodos activos y ha sido utilizada para control y supervisión en procesos biológicos continuos para el tratamiento de aguas (Spanjers et al., 1998), lo cual ha sido verificado para el caso de estudio por medio de simulaciones con cambios en parámetros y condiciones iniciales. Los cambios en la dinámica del proceso se reflejan directamente en el señal de la tasa de respiración de la fase de llenado y reacción, como se muestra en la Figura (3.4).

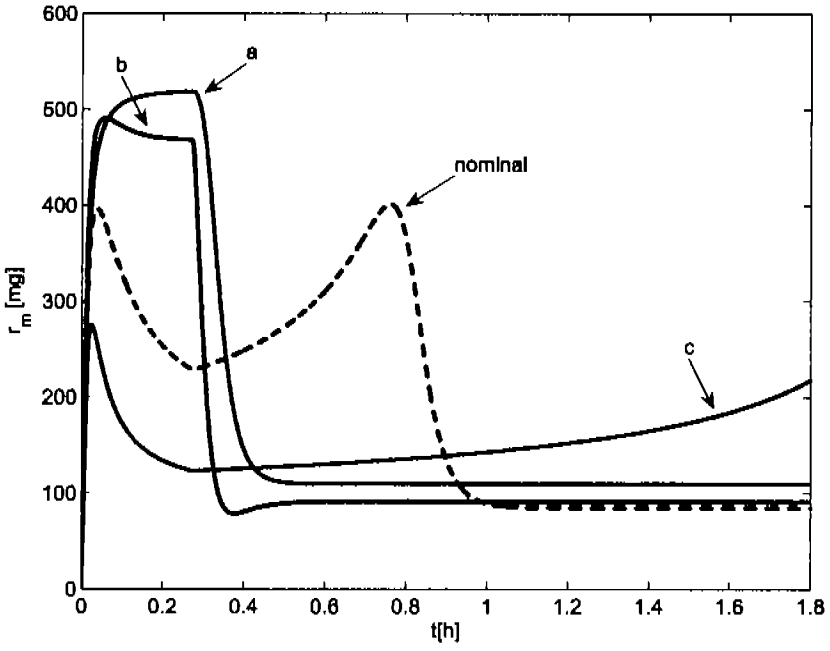


Figura 3.4: Perfiles de respiración (nominal discontinuo, a con $c_{s,in} - 30\,\%$ y $m_x(0) + 30\,\%$, b con $c_{s,in} - 30\,\%$, $m_x(0) + 30\,\%$, $K_s - 30\,\%$ y $K_i + 50\,\%$, c con $c_{s,in} + 30\,\%$, $m_x(0) - 30\,\%$; todas las desviaciones de valores nominales (Tabla 2.2).

El equipo comercial para medir la tasa de respiración representaría un costo y factor de mantenimiento adicional, y además no puede proporcionar una estimación continua de la tasa de respiración *in situ*. Sin embargo, como se muestra en la Figura 3.3 r_m es monitoreable y por lo tanto puede estimarse en línea.

Por un lado existe la posibilidad de calcular r_m directamente del balance de oxígeno en la Ec. (2.2) y la medición de oxígeno disuelto. No obstante, esta estimación es muy sensible con respecto a ruido en la medición de oxígeno disuelto. Como alternativa, se puede utilizar un filtro digital en lazo cerrado o un observador para reducir el efecto de ruido en la medición de oxígeno a la estimación de r_m. Eso ha sido propuesto para procesos en continuo por Marsili-Libelli y Vaggi (1997) y Lindberg y Carlson (1996). Para procesos en lotes, Yoong et al. (2000) describen un método para obtener la tasa de respiración *in situ*, pero no continuamente. Sin embargo, éstos métodos suponen un volumen constante, lo cual no es válido para la operación que se considera en el caso de estudio, ya que hay un cambio de volumen mientras se inicia la reacción.

Para resolver el problema de la estimación, se propone usar un observador de tiempo discreto que

estima la tasa de respiración r_m, similar a la propuesta en Wimberger y Verde (2005) para el modelo de concentraciones. Ec. (2.2) se puede discretizar usando un retenedor de orden cero (ZOH, por sus siglas en ingles) propuesto por Åström y Wittenmark (1984), lo cual arroja como resultado una forma discreta con respecto al tiempo descrito por:

$$m_o(k+1) = \theta_1\, m_o(k) - \theta_2\, r_m(k) + \theta_2\, K_{la}\, m_{o,sat} + \theta_2\, q_{in}\, c_{o,in} \tag{3.15}$$

donde

θ_1 $= e^{-K_{la}\, h}$

θ_2 $= \frac{1}{K_{la}}(1-\theta_1)$

h es el tiempo entre muestras

k es el paso normalizado

Para la transformación se supone que la tasa de respiración r_m cambia mucho más lentamente que m_o y por lo tanto se puede describir con:

$$r_m(k+t) = r_m(k) + n(k) \quad \text{para } 0 < t < 1 \tag{3.16}$$

donde $n(k)$ es un ruido blanco.

De las Ec. (3.15) y (3.16) se puede obtener un modelo aumentado descrito por:

$$\begin{bmatrix} m_o(k+1) \\ r_m(k+1) \end{bmatrix} = A \begin{bmatrix} m_o(k) \\ r_m(k) \end{bmatrix} + B \begin{bmatrix} m_{o,sat}(k) \\ q_{in}\, c_{o,in} \end{bmatrix} + C\, n(k) \tag{3.17}$$

donde

$$A = \begin{bmatrix} \theta_1 & \theta_2 \\ 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} \theta_2\, K_{la} & \theta_2 \\ 0 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

La Ec. (3.17) permite la observación de r_m con un observador de orden total o reducido.

En este trabajo se propone usar la Ec. (3.18), la cual se basa en la estructura de un observador descrito por (Åström y Wittenmark, 1984). Este observador utiliza la información disponible hasta el momento k para proporcionar una estimación de la tasa de respiración r_m en el momento k, lo cual se indica por $(k\backslash k)$.

$$\begin{bmatrix} \hat{m}_o(k\backslash k) \\ \hat{r}_m(k\backslash k) \end{bmatrix} = (I - KC)\left(A \begin{bmatrix} m_o(k-1) \\ r_m(k-1) \end{bmatrix} + B \begin{bmatrix} m_{o,sat}(k-1) \\ q_{in}(k-1)\, c_{o,in} \end{bmatrix} \right) + Ky(k) \tag{3.18}$$

donde K se elige tal que el observador es asintóticamente estable (Åström y Wittenmark, 1984).

La robustez del observador con respecto a incertidumbres en la condición inicial de r_m se muestra en la Figura 3.5: después de 0.01 horas el error es practicamente zero.

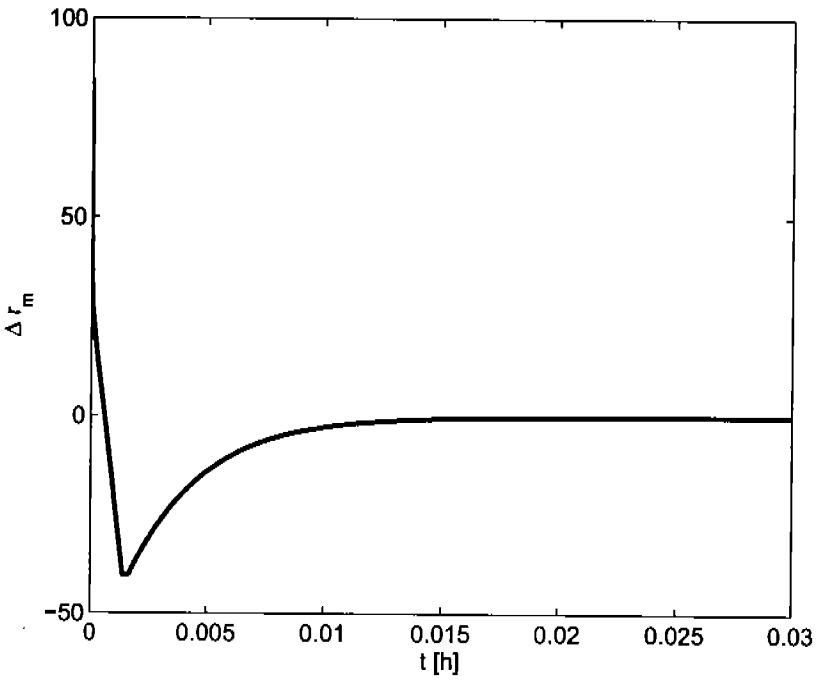


Figura 3.5: Error con respecto al tiempo en la estimación de $\hat{r}_m$, entre la estimación con $\hat{r}_m(0) = 0$ y r_m obtenida de la simulación (i.e. valor correcto).

Factores importantes para la estimación de $\hat{r}_m$

La estimación de la tasa de respiración $\hat{r}_m$ depende de varios factores que se deben de considerar en la práctica.

Considerando que la estimación de $\hat{r}_m$ depende de la medición del oxígeno disuelto en el reactor, se debe evaluar la dinámica del sensor. Si la dinámica del sensor de concentración de oxígeno disuelto es comparable a la dinámica del proceso que se quiere observar, el retraso en la respuesta del sensor se tiene que tomar en cuenta. Este problema se discute con detalle en Wimberger y Verde (2005).

Otro factor importante que influye la estimación de $\hat{r}_m$ es la concentración de oxígeno al punto de saturación $c_{o,sat}$ utilizado para calcular $m_{o,sat}$. El valor de $c_{o,sat}$ se define como el máximo de concentración de oxígeno en la fase líquida, lo cual es equivalente a la solubilidad de oxígeno en soluciones acuosas. La solubilidad es una función que depende de:

- la temperatura;
- la presión parcial de oxígeno; y
- la salinidad de la solución.

Aunque la influencia de la temperatura ha sido estudiada y verificada por Vogelaar et al. (1996), en el área de tratamiento de agua todavía es práctica común utilizar tablas de solubilidad de oxígeno. Adicionalmente, modelos convenientes para la solubilidad de oxígeno en soluciones acuosas han sido descrito por Tromans (1999). Estos se pueden utilizar para calcular el $c_{o,sat}$ a partir de una medición

de temperatura y una estimación de la presión parcial del oxígeno, utilizando la ley de Henry y una estimación de la salinidad adentro del reactor.

Finalmente, el coeficiente de la transferencia de oxígeno K_{la} puede cambiar lentamente en función del tiempo, lo cual hay que considerar en el momento de la estimación. Por un lado los métodos para la estimación del K_{la} en procesos bioquímicos presentadas en la revisión de Gogate y Pandit (1999) no aplican en el caso de reactores de tratamiento de agua, donde comúnmente solo se puede contar con un único sensor de concentración de oxígeno disuelto, como en el caso de estudio. Por otro lado, los métodos propuestos por Lindberg y Carlson (1996) y Marsili-Libelli y Vaggi (1997) solo aplican en casos de procesos continuos de tratamiento de agua, bajo las suposiciones que el volumen es constante y que el flujo de aire se puede controlar. Sin embargo, ninguna de estas dos suposiciones son válidas en el caso de estudio. Por lo tanto en este trabajo se propone:

1. Evaluar el efecto de una desviación en el parámetro K_{la},; y
2. diseñar un método para la estimación de K_{la} *in situ*.

Estos dos propuestas se elaboran con detalle a continuación y en la Sección 3.4.1.

3.2.2.3. Análisis de sensibilidad de la tasa de respiración

El primer paso del método propuesto en la Sección 3.2.2.1 es la definición de un conjunto de parámetros y condiciones iniciales en las cuales se consideran desviaciones. En el proceso del caso de estudio, la dinámica de la tasa de respiración (Fig. 3.4) se ve afectada por:

- la condición inicial de la masa de los lodos activos $m_x(0)$, la cual puede variar por cambios en la población de los lodos activos, o purga;
- la concentración de sustrato en el flujo de entrada $c_{s,in}$, la cual puede cambiar por el origen del agua tratada y ciclos de producción o uso;
- el coeficiente de la transferencia de oxígeno K_{la}, el cual se discutió al final de la sección anterior;
- el comportamiento de crecimiento de los lodos activos con sustratos que inhiben su actividad metabólica, en particular los parámetros de la Ec. (2.3), los cuales pueden variar por cambios poblacionales en los lodos activos:
  - K_i, el coeficiente de inhibición;
  - K_s, el coeficiente de media saturación; y
  - μ_0, el coeficiente de la tasa específica de crecimiento.

Por lo tanto, el análisis de sensibilidad se concentra en el conjunto de parámetros y condiciones iniciales

$$P_{I,F} = \{m_x(0), c_{s,in}, K_{la}, K_i, K_s, \mu_0\} \tag{3.19}$$

suponiendo, que son constantes durante una fase de llenado y reacción.

Como ejemplo, la función de sensibilidad dependiente del tiempo para la variable r_m con respecto a la condición inicial $m_x(0)$ es:

$$\sigma(t, m_x(0)_0) \triangleq \frac{\partial r_m}{\partial m_x(0)} \mid_{m_x(0)=m_x(0)_0} \tag{3.20}$$

donde $m_x(0)_0$ es el valor nominal de la condición inicial de los lodos activos (Tabla 2.2). Esta función se puede normalizar con respecto al valor de la tasa de respiración:

$$\overline{\sigma} = \frac{\sigma(t, m_x(0)_0)}{r_m} \tag{3.21}$$

Las funciones de sensibilidad normalizadas de r_m con respecto a la desviación en $m_x(0)$, $c_{s,in}$, K_s, K_i, μ_0 y K_{la} y con respecto al tiempo durante la fase de llenado y reacción se muestran en la Figura 3.6.

La evolución de las funciones de sensibilidad normalizadas presentadas en la Figura 3.6 se puede comparar cualitativamente para encontrar puntos y regiones donde existen diferencias significativas en tiempo y amplitud de efectos causados por la desviación en diferentes parámetros. En particular, esta comparación arroja las siguientes observaciones:

1. Las desviaciones en $c_{s,in}$ y $m_x(0)$ afectan el comportamiento de r_m de manera similar. Sin embargo, el efecto de desviación en estos parámetros es distinto a los efectos causados por desviaciones en K_s, K_i y μ_0;

2. La desviación de K_{la} afecta a r_m durante toda la fase de llenado y reacción. En particular modifica el valor del mínimo, del segundo máximo y el valor de la respiración endógena r_e al final de la fase;

3. La desviación de K_s afecta a r_m al principio de la fase y en el segundo máximo;

4. La desviación de K_i afecta fuertemente al segundo máximo de la señal;

5. Los efectos de las desviaciones en K_s y K_i en el segundo máximo son opuestos;

6. Las desviaciones en μ_0 y K_i afectan el comportamiento de r_m de manera muy similar.

Estas observaciones indican que es posible diseñar un algoritmo de DAF utilizando la tasa de respiración r_m.

3.3. Formulación del problema diagnóstico

Con base en las observaciones que arroja la comparación de las funciones de sensibilidad del análisis en Sección 3.2.2.3 se definen dos subconjuntos del conjunto $P_{I,F}$ (3.19):

1. P_I el subconjunto de parámetros y condiciones iniciales de las cuales toda desviación del valor nominal se define como incertidumbre;

2. P_F el subconjunto de parámetros y condiciones iniciales de las cuales la desviación del valor nominal a partir de un cierto tamaño se define como falla.

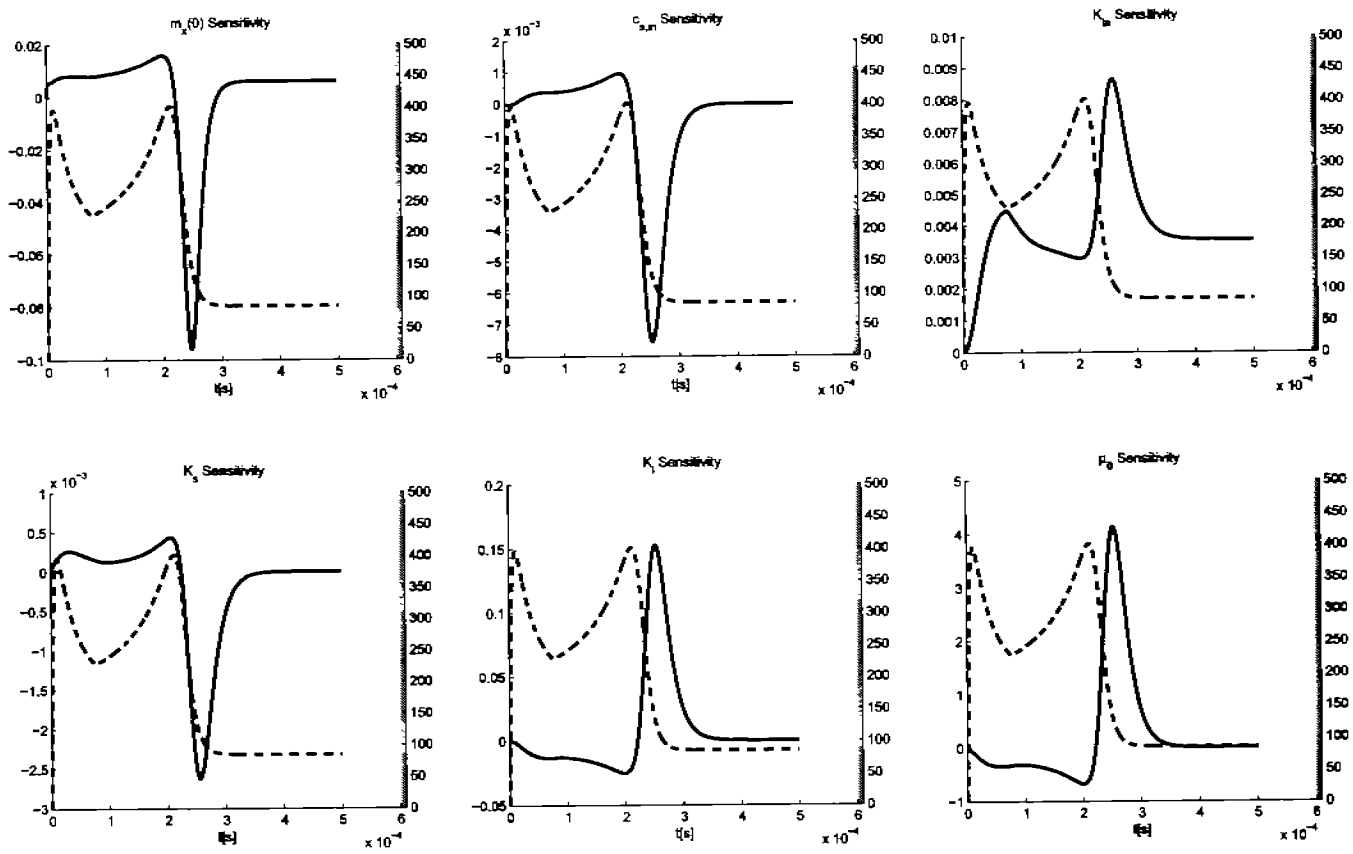


Figura 3.6: Funciones de sensibilidad normalizadas de r_m (línea discontinua) con respecto a los parámetros y condiciones iniciales de interés

Primero, el conjunto de parámetros y condiciones iniciales de las cuales toda desviación del valor nominal se define como incertidumbre. En particular, la condición inicial de la masa de lodos activos $m_x(0)$ y la concentración de sustrato en el flujo de entrada $c_{s,in}$ se definen por las condiciones de operación de un SBR específico. En procesos como el proceso del caso de estudio, comúnmente son inciertos y los dos varían entre ciclos por la variación en flujo y composición de las aguas residuales. Sin embargo, dados unos datos de operación, se define el intervalo dado por

$$p_{nominal} \pm \Delta \tag{3.22}$$

donde Δ es un porcentaje que refleja la incertidumbre que se espera en un caso especifico del valor nominal. Por lo tanto el subconjunto a tratar está dado por:

$$P_I = \{m_x(0), c_{s,in}\} \tag{3.23}$$

Además, dado el hecho que las desviaciones en $c_{s,in}$ y $m_x(0)$ afectan el comportamiento de r_m de manera distinta a los otros parámetros, se puede suponer que es posible para un algoritmo de DAF manejarlos como incertidumbres.

El otro subconjunto es formado por parámetros y condiciones iniciales de los cuales la desviación del valor nominal a partir de un cierto tamaño se define como falla:

$$P_F = P_{I,F} \backslash P_I = \{K_{la}, K_i, K_s, \mu_0\} \tag{3.24}$$

En éste caso, se supone que la desviación del valor nominal hasta un cierto tamaño es resultado de un error en la identificación o la estimación de los parámetros o condiciones iniciales, que todavía debe clasificarse como condición normal:

$$p_{nominal} \pm \epsilon; \epsilon > \Delta \tag{3.25}$$

donde Δ es un porcentaje que refleja la incertidumbre en el valor nominal y ϵ es un porcentaje que refleja una desviación que se define como falla.

En el caso de K_s, K_i las observaciones del análisis de sensibilidad indican que es posible para un algoritmo de DAF detectar y aislar fallas si no ocurren al mismo tiempo. Esto no es el caso para desviaciones en μ_0 y K_i, las cuales afectan el comportamiento de r_m de manera muy similar. Por lo tanto, se supone que desviaciones en μ_0 se detectan como fallas, pero no se pueden aislar de desviaciones en K_i y serán reportados como fallas en K_i .

En el caso del coeficiente de la transferencia de oxígeno K_{la} , la Figura 3.6 y la segunda observación correspondiente indican que una desviación afecta la evolución de la señal r_m durante toda la fase. Por lo tanto:

- se requiere el conocimiento de K_{la} con mucha certidumbre para poder detectar fallas en en K_i, K_s y μ_0;
- no se puede considerar la detección de fallas en el parámetro K_{la} a partir del señal r_m al mismo tiempo que fallas en K_i, K_s y μ_0.

Dado los resultados de la evaluación de viabilidad para el enfoque analítico, el enfoque con base en señales, y las consideraciones anteriores, se propone utilizar el enfoque con base en señales y, en particular, un método basado en la señal de la respiración (Sección 3.2.2.2). La excepción es el parámetro K_{la}, para el cual se requiere un método alternativo, el cual:

1. no utiliza la tasa de respiración r_m; y

2. permite garantizar que la estimación de r_m no se ve afectada por desviaciones en este parámetro.

Finalmente, se hace notar, que cambios lentos en los parámetros entre la fase de llenado y reacción de diferentes ciclos se detectaran en ciclos subsiguientes.

3.4. Solución al problema diagnóstico

3.4.1. Estimación y monitoreo del coeficiente de transferencia K_{la}

El conocimiento del valor del coeficiente de la transferencia de oxígeno K_{la}, es importante para una buena estimación de la tasa de respiración y, consecuentemente, para la viabilidad de un diagnóstico basado en la tasa de respiración r_m. Por lo tanto, se propone un método para la estimación de K_{la} *in situ*, aprovechando las mediciones del sensor de oxígeno disuelto durante la fase de re-aireación (Figura 2.4) bajo las siguientes suposiciones:

1. El parámetro K_{la} cambia lentamente si el flujo de aire es constante (Lindberg y Carlson, 1996);

2. La tasa de respiración es aproximadamente constante bajo la ausencia del sustrato (respiración endógena, r_e);

3. La temperatura es constante durante la fase de re-aireación, que es de aproximadamente 30 minutos en el caso de estudio.

La identificación del parámetro K_{la} proporciona una estimación que a su vez se puede utilizar en la subsiguiente fase de llenado y reacción para la estimación de r_m. Con base en las suposiciones anteriores, se puede obtener un modelo discreto del balance de oxígeno en la Ec. (2.2) utilizando un retenedor de orden cero y, reducirlo a

$$\Delta O(k) = \theta \, \Delta O(k-1) \tag{3.26}$$

donde ΔO es la diferencia en dos muestras subsiguientes de la concentración de oxígeno disuelto en el tanque, $\Delta O(k-i) = O(k-i) - O(k-(i+1))$ y $\theta = e^{-K_{la}\, h}$.

La Ec. (3.26) describe el comportamiento de la concentración del oxígeno disuelto en una forma incremental (Åström y Wittenmark, 1984, p.181), la cual no depende de la tasa de respiración o de la concentración de oxígeno en el punto de saturación. Utilizando en Ec. (3.26) se propone la implementación de dos algoritmos de identificación para el parámetro K_{la}:

1. una regresión normal de mínimos cuadrados (OLS; por sus sigla en ingles), usando todas las muestras obtenidas durante la fase de re-aireación, lo cual es un método estándar para la identificación de parámetros fuera de línea (Söderström y Stoica, 1989, Sec. 4.1); y

2. un filtro de Kalman para la identificación de parámetros, un método recursivo que funciona en línea (Söderström y Stoica, 1989, p.325).

Ambos pueden ser aplicados si la re-aireación empieza con la condición inicial $O(0) << O_{sat} - r_e$ y acaba cerca de las condiciones del estado permanente ($\frac{dO}{dt} \cong 0$, ó $O \cong O_{sat} - r_e$).

Los métodos recursivos, en este caso el filtro de Kalman, tienen dos ventajas sobre el OLS:

1. la estimación del parámetro puede mejorar en ciclos subsiguientes, si se utiliza el valor estimado $\hat{K}_{la}$ de un ciclo como punto de arranque para el filtro en el ciclo subsiguiente; y
2. la estimación del parámetro requiere menos recursos de cómputo.

Por lo tanto, se propone el uso del filtro de Kalman para la estimación de K_{la} en el caso de estudio. Sin embargo, como alternativa existe la posibilidad de aplicar una regresión recursiva de minimos cuadrados (RLS; por sus siglas en ingles), la cuál arrojaria resultados equivalentes a los del filtro de Kalman.

Se hace notar, que la estimación del parámetro K_{la} durante ciclos subsiguientes forma una serie temporal univariada, que permite la detección de cambios por medio de un método estadístico como el que se presenta en la Sección 3.1 del articulo de Venkatasubramanian et al. (2003c): se diseña una función estadistica $g(t)$ de la observación $x(t)$ y la decisión de cambio se basa en una comparación de $g(t)$ con un valor umbral c. Donde $g(t)$ puede ser el promedio, un promedio móvil, la varianza etc. y $x(t)$ es la estimación del parámetro K_{la}.

3.4.2. Diagnóstico de fallas con las características extrínsecas de los lodos activos

3.4.2.1. Metodología del diagnóstico de fallas

En el caso de estudio se carece de un conocimiento heurístico que vincule el comportamiento de los lodos activos con la evolución de la señal r_m, en condiciones normales y anormales. Sin embargo, es posible obtener este conocimiento en forma de un mecanismo de inferencia que sirva para el diagnóstico de comportamientos, a partir de datos e información del diagnóstico del proceso (Venkatasubramanian et al., 2003c; Isermann, 2000), los cuales provienen de mediciones y la bitácora de operación.

El análisis de sensibilidad presentado en Sección 3.2.2.3 arroja como resultado adicional un conjunto de características que describen un patrón en la señal:

$$\mathcal{F} = [f_1, f_2, ..., f_n] \tag{3.27}$$

Bajo la suposición que desviaciones en los parámetros del modelo se reflejan en cambios en los valores de las características $f_i \in \mathcal{F}$ (i.e. diferentes patrones) y, por lo tanto, permite la detección y el aislamiento de fallas, se propone utilizar un enfoque basado en la extracción y clasificación de estas características.

Cuando existen datos del proceso e información del diagnóstico correspondiente que identifica casos normales y casos con presencia de fallas, los algoritmos de aprendizaje de máquina pueden extraer modelos de clasificación que representan implícitamente o explícitamente el conocimiento que asocia síntomas observados con fallas (Witten y Frank, 2000). Estos algoritmos proporcionan un mecanismo de inferencia para diagnosticar comportamientos futuros del proceso. La metodología que resulta de los pasos de extracción y clasificación para obtener un diagnóstico, se presenta de forma visual en la Figura 3.7.

En el caso de estudio al momento aún no existen suficientes datos del proceso con un diagnóstico asociado que identifique claramente casos normales y con fallas. Por lo tanto, se propone como alternativa utilizar el modelo analítico del proceso (Ec. 2.2) para generar datos, tomando en cuenta las incertidumbres consideradas y todos los casos de falla definidos. De esta manera, el esquema del enfoque de DAF general presentado en la Figura 3.7 se traduce al caso de estudio con el esquema particular presentado en la Figura 3.8. En las dos secciones subsiguientes se presentan detalles del conjunto de características $\mathcal{F}$, su extracción de la señal r_m y la parte de la clasificación.

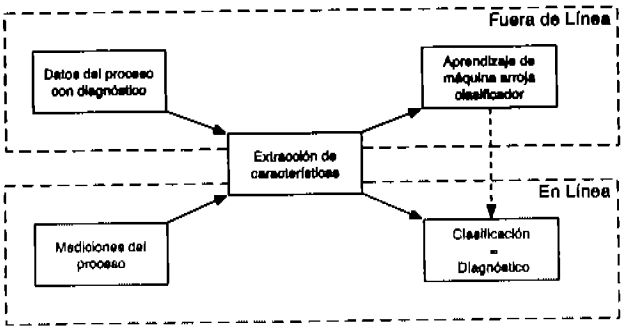


Figura 3.7: Enfoque de DAF basado en extracción y clasificación de características de señales

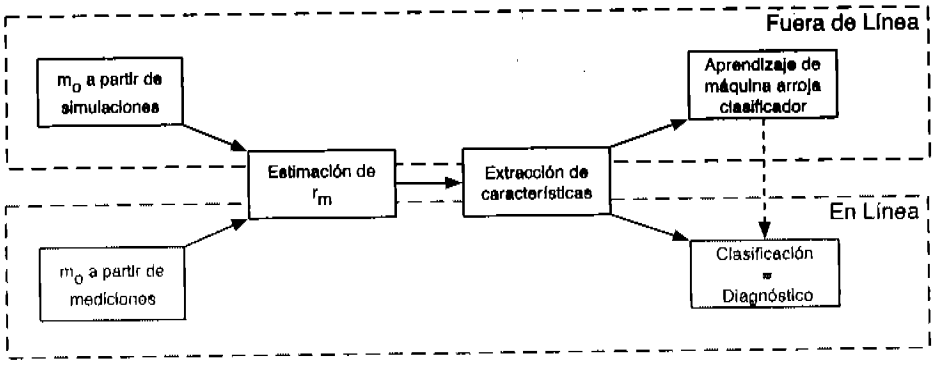


Figura 3.8: Metodología de diagnóstico para el caso de estudio

3.4.2.2. Características de la tasa de respiración

Con base en los resultados y observaciones de la evaluación de la viabilidad de un diagnóstico presentado para la señal de respiración (Sección 3.2.2.3), se propone extraer las siguientes características de la señal de respiración $\hat{r}_m$:

- $r_{max,1}$ y $r_{max,2}$ las dos máximas;
- r_{min} el mínimo entre los dos máximos;
- $t_{max,1}$ el tiempo para llegar a $r_{max,1}$; y
- las propiedades geométricas del triángulo $(a, b, c, \alpha, \beta, \gamma)$ definido por $r_{max,1}, r_{max,2}$ y r_{min}.

Adicionalmente, hay dos características que se pueden extraer de r_m, las cuales se seleccionaron con base en la experiencia de expertos humanos del área de bioprocesos:

- r_e la tasa de respiración endógena; y
- r_{tot} el total de la respiración metabólica definido por

$$r_{tot} = \int_0^{t_{react}} (r_m - r_e)\, dt$$

El conjunto de características es por lo tanto:

$$\mathcal{F} = \{t_{max,1}, r_{max,1}, r_{max,2}, r_{min}, a_t, b_t, c_t, \alpha_t, \beta_t, \gamma_t, r_e, r_{tot}\} \tag{3.28}$$

Todas estas características, con excepción de $t_{max,1}$, están indicadas en la Figura 3.9.

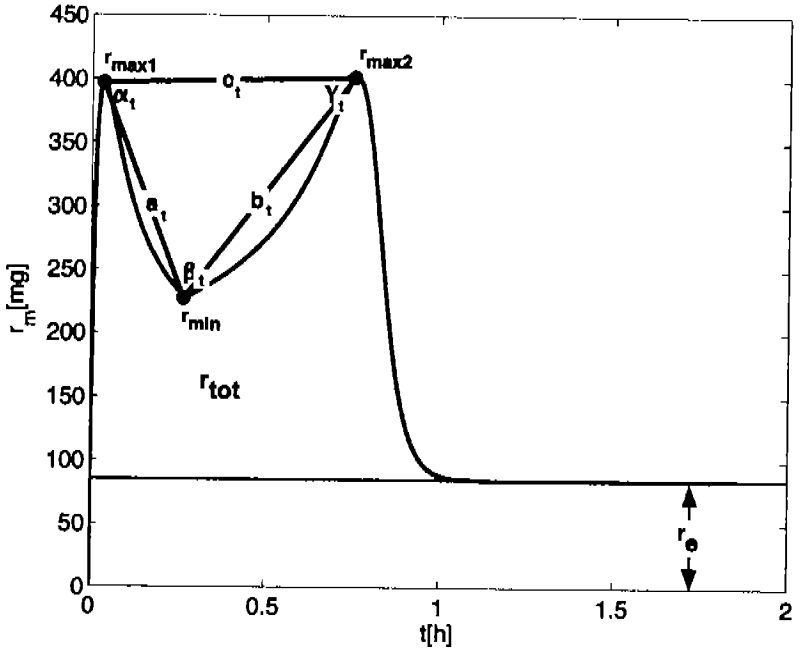


Figura 3.9: Características de r_m en el patrón nominal de la señal

Con excepción de r_e las características se extraen y calculan a partir de una normalización considerando el mínimo y el máximo de la señal r_m.

Cabe hacer notar que existen patrones cualitativamente diferentes al patrón nominal de la Figura 3.4, los cuales tienen un diagnóstico conocido para el experto humano del área de bioprocesos. Estos casos son:

1. patrones similares a las señales a y b de la Figura 3.4 corresponden al caso de ausencia de inhibición y, por lo tanto, un comportamiento normal del proceso; y

2. un patrón similar a la señal c de la Figura 3.4 corresponde al caso de la degradación incompleta del 4-CF en la fase de llenado y reacción. Este caso es el peor escenario, ya que sin intervención del operador o del sistema de supervisión, puede causar la emisión de la sustancia tóxica al ambiente.

Estos casos se pueden detectar con el algoritmo de extracción propuesto en esta sección, sin embargo, no se pueden diagnosticar con la clasificación propuesta en la siguiente sección.

3.4.2.3. Clasificación

La detección de fallas en uno de los parámetros K_s o K_i corresponde a una clasificación de las características extraídas de r_m en dos clases:

1. *normal*: la cual indica un comportamiento normal del proceso;
2. *anormal*: la cual indica un comportamiento en presencia de una falla en uno de los dos parámetros.

Para la identificación del tipo de falla, se requieren dos clases adicionales:

1. K_s: la cual indica la presencia de una falla en el parámetro K_s;
2. K_i: la cual indica la presencia de una falla en el parámetro K_i.

Finalmente, debido a la información disponible de la señal, se considera que no se puede determinar el tamaño del cambio, pero si se puede determinar la dirección en la cual se desvió el parámetro. Este diagnóstico requiere de 4 clases adicionales:

1. $K_s \downarrow$: la cual indica la presencia de una falla en el parámetro K_s, causado por una desviación mayor a $-\epsilon\,\%$ del valor nominal;
2. $K_s \uparrow$: la cual indica la presencia de una falla en el parámetro K_s, causado por una desviación mayor a $+\epsilon\,\%$ del valor nominal;
3. $K_i \downarrow$: la cual indica la presencia de una falla en el parámetro K_i, causado por una desviación mayor a $-\epsilon\,\%$ del valor nominal;
4. $K_i \uparrow$: la cual indica la presencia de una falla en el parámetro K_i, causado por una desviación mayor a $+\epsilon\,\%$ del valor nominal;

Por lo tanto, se propone la generación de tres diferentes clasificadores, los cuales deben poder clasificar un conjunto de características (Ec. 3.28) extraídas de r_m en una de las clases del conjunto correspondiente al clasificador:

1. *{normal, anormal}*: conjunto de clases del primer clasificador, asociado al caso de detección;
2. *{K_s, K_i, normal}*: conjunto de clases del segundo clasificador, asociado al caso de detección y aislamiento del parámetro que desvió;
3. *{$K_s \downarrow$, $K_s \uparrow$, $K_i \downarrow$, $K_i \uparrow$, normal}*: conjunto de clases del tercer clasificador, asociado al caso de detección y aislamiento del parámetro que desvió y, la dirección de la desviación.

En general, los modelos de clasificación o clasificadores, se pueden categorizar en dos grupos principales:

1. *Caja negra*: el conocimiento es implícito, como por ejemplo en redes neuronales;
2. *Caja blanca*: el conocimiento es explicito, como por ejemplo, en sistemas expertos basados en reglas o árboles de decisión.

Ya que en algunos casos puede ser interesante tener acceso al conocimiento extraído por un algoritmo de aprendizaje de máquina de los datos, se propone utilizar un algoritmo de cada categoría. Esto, al mismo tiempo permite la comparación de resultados y la posibilidad de elegir el mejor para el caso de estudio en particular.

Las implementaciones de los algoritmos usados en este trabajo forman parte de una herramienta de aprendizaje de máquina denominado WEKA (Witten y Frank, 2000). Estos algoritmos son:

1. *J48*: una implementación del algoritmo C4.5 para el aprendizaje de árboles de decisión. Este algoritmo requiere poco tiempo para el aprendizaje y produce un modelo de caja blanca, representado por un árbol de decisiones binarias; y

2. *MLP*: un algoritmo que construye un perceptron de múltiples capas (MLP por sus siglas en ingles) y lo entrena por medio de propagación inversa (back-propagation). En este caso se propone el uso de un MLP con una sola capa escondida entre la capa de entrada y la capa de salida. El número de nodos de entrada, nodos de salida y de nodos en la capa intermedia es igual a:

   a) a, el número de características en el conjunto $\mathcal{F}$;

   b) c, el número de clases de diagnóstico; y

   c) $(a + c)/2$, respectivamente.

   Este algoritmo requiere un orden de magnitud mayor de tiempo para el aprendizaje (e.g. entrenamiento) y produce un modelo de caja-negra.

3.5. Validación de la solución

3.5.1. Estimación y monitoreo del coeficiente de transferencia K_{la}

La implementación de los dos algoritmos de identificación propuestos en la Sección 3.4.1 han sido verificados exitosamente en simulación, y con datos experimentales (WP1 Process Experiments, 2004). En particular, la Figura 3.10 presenta un ejemplo de la evolución del oxígeno disuelto durante la fase de re-aireación.

El valor $\hat{K}_{la,A} = 0{,}00461[s^{-1}]$ es el resultado de la identificación con el filtro Kalman, cuando éste se inicia en el mismo momento en el que se enciende el sistema de aireación (usando la covarianza inicial $P(0) = 100$ y $\theta(0) = 0$), $\hat{K}_{la,B} = 0{,}00464[s^{-1}]$ es el resultado obtenido cuando el algoritmo se inicia en el instante que el sistema de aireación llego al estado permanente.

Este último caso permite obtener un mejor resultado para el valor del parámetro, ya que la primera parte de la curva de re-aireación es más critica para la identificación y se ven afectados por el retraso del sistema de aireación para llegar al estado permanente. Los resultados son comparables con el valor $K^*_{la} = 0{,}00467[s^{-1}]$ reportado por Betancur et al. (2004) para los mismos datos.

3.5.2. Diagnóstico de fallas con las características extrínsecas de los lodos activos

La realización del método propuesto en la Sección 3.4.2, que se integra por la extracción y la clasificación de características, se evaluó con dos objetivos principales:

1. estudiar el rendimiento del diagnóstico nominal, tomando en cuenta las incertidumbres definidas;

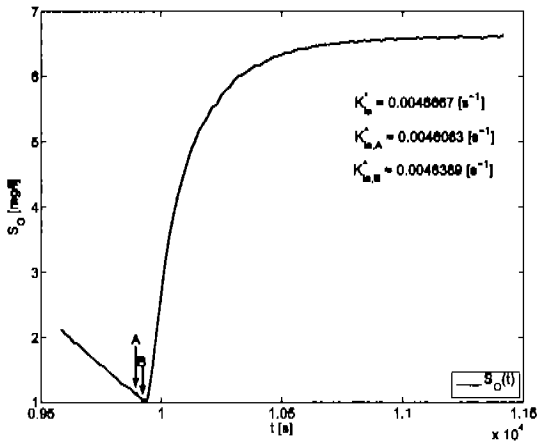


Figura 3.10: Concentración de oxígeno durante la fase de re-aireación y valores del parámetro K_{la} identificado dos métodos (A es el instante cuando empieza la fase, B es el instante cuando se puede asumir que el sistema de aireación llegó al estado permanente).

2. evaluar la robustez del diagnóstico, con respecto a casos no considerados para la generación de los clasificadores.

Para la evaluación, se ejecutó un gran número de simulaciones con el modelo analítico del proceso (Sección 2.2.5), para la generación de conjuntos de datos que constan de:

1. las características dc la señalde respiración $\mathcal{F}$ (Sección 3.4.2.2); y
2. la información que representa el diagnóstico, en forma de una de las clases definidas en la Sección 3.4.2.3.

En las simulaciones, de acuerdo con la Sección 3.3, se tomaron en cuenta incertidumbres en las siguientes condiciones iniciales y parámetros del modelo:

1. desviaciones en $m_x(0)$ y $c_{s,in}$, adentro del intervalo dado por $\pm 25\,\%$ del valor nominal (i.e. $\Delta = 25$); y
2. desviaciones en K_i y K_s, adentro del intervalo dado por $\pm 5\,\%$ del valor nominal (i.e. $\Delta = 5$).

Tomando en cuenta la incertidumbre en los valores de $m_x(0)$ y $c_{s,in}$, y con el objetivo de tener una muestra representativa de posibles condiciones, se tomaron 200 muestras aleatorias con una distribución uniforme. Está distribución se utilizó, dado que en el caso de estudio:

1. es representativa para los datos experimentales disponibles (WP1 Process Experiments, 2004); y
2. refleja el estado de conocimiento actual sobre el proceso;

conforme con la sugerencia hecha para la selección de distribuciones para llevar a cabo simulaciones Monte-Carlo por Robert y Casella (2004).

Para cada conjunto de condiciones nominales muestreadas aleatoriamente para la simulación, se generaron los siguientes 40 conjuntos de datos para diferentes desviaciones en K_i y K_s:

1. *comportamientos normales*:
   por medio de desviaciones de $\pm 5\,\%$ del valor nominal de K_i o K_s en incrementos de $1\,\%$; y

2. *comportamientos anormales*:
   por medio de desviaciones de $\pm 50\,\%$ del valor nominal de K_i o K_s en incrementos de $10\,\%$.

Esto corresponde a aproximadamente 8000 señales(i.e. 200 por 40) de respiración generados a partir de simulaciones en una sola corrida, para las cuales se registraron las características y la clase diagnóstica correspondiente.

3.5.2.1. Rendimiento del diagnóstico nominal

Para la evaluación del rendimiento del diagnóstico nominal, se dividieron los conjuntos de datos de manera aleatoria en dos subconjuntos:

1. el subconjunto de entrenamiento con aproximadamente $66\,\%$ del número total; y

2. el subconjunto de prueba con aproximadamente $33\,\%$ del número total.

El conjunto de datos de entrenamiento se utilizó para la generación de los seis clasificadores descritos en la Sección 3.4.2.3:

- uno únicamente para la detección (2-clases);
- uno para el diagnóstico del parámetro que se desvía (3-clases); y
- uno para el diagnóstico del parámetro que se desvía y la dirección en la cual se desvía (5-clases);

de cada uno de los algoritmos de aprendizaje de maquina seleccionados (J48 y MLP).
El conjunto de prueba se utilizó para evaluar el diagnóstico generado por los clasificadores para casos que no se utilizaron en el entrenamiento. Este procedimiento es conocido como validación cruzada.
La comparación del promedio de clasificaciones de datos de 10 corridas de simulaciones se presenta en la Figura 3.11. Se muestra que, a pesar de incertidumbres significativas en $m_x(0)$ y $c_{s,in}$, la metodología propuesta es capaz de diagnosticar correctamente alrededor de $95\,\%$ de todos casos con los dos tipos de clasificadores.

3.5.2.2. Robustez del diagnóstico

Para la evaluación de la robustez del diagnóstico ante incertidumbres o casos no considerados, se generaron conjuntos de datos de prueba con casos no utilizados o considerados para el entrenamiento de los clasificadores que arrojan como resultado el diagnóstico:

1. *Conjunto 1*: ≈ 300 conjuntos de características donde
   $m_x(0) = m_x(0)|_{nom} \cdot \Delta_1$
   $c_{s,in} = c_{s,in}|_{nom} \cdot \Delta_1$
   y
   $K_s = K_s|_{nom} \cdot \Delta_2$
   o
   $K_i = K_i|_{nom} \cdot \Delta_2$
   y $0{,}75 \leq \Delta_1 \leq 1{,}25$, $0{,}5 \leq \Delta_2 \leq 1{,}5$. Este conjunto evalúa la robustez de la clasificación para un gran número de casos generados aleatoriamente.

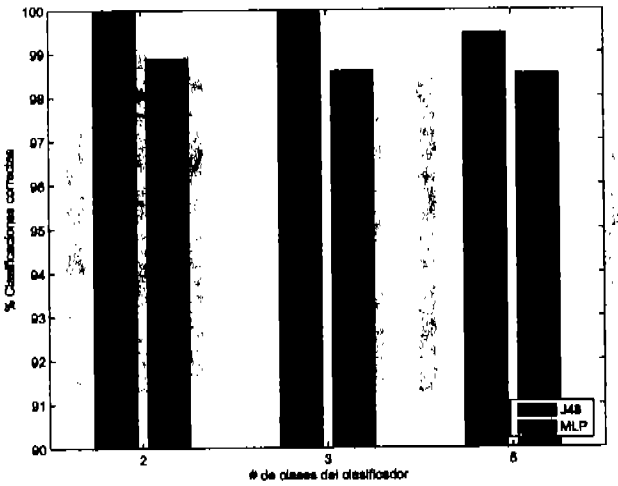


Figura 3.11: Comparación del rendimiento del diagnóstico, visualizado entre 90 % y 100 % de clasificaciones correctas.

2. *Conjunto 2:* similar al conjunto 1, pero incluyendo una desviación en las características del tipo:

$$f_i = f_i(1 + 0{,}01 \cdot \xi_i)$$

donde $f_i \in \mathcal{F}, i = 1...n$, ξ_i es una variable aleatoria con una distribución uniforme ($0 \leq \xi_i \leq 1.$), y $i = 1, 2, 3, 4, 11$ los características independientes de la señal. Éste conjunto evalúa la robustez del diagnóstico con respecto a incertidumbres en los elementos del conjunto de características $\mathcal{F}$. Cabe hacer notar, que tendría un resultado equivalente si se hubiera añadido ruido a la señal r_m. Sin embargo, esta opción abre la posibilidad de estudiar el efecto de desviaciones en cada característica f_i al diagnóstico por separado.

3. *Conjunto 3*: ≈ 200 conjuntos de características donde
$K_s = K_s|_{nom} \cdot \Delta_3$
$K_i = K_i|_{nom} \cdot \Delta_3$
y $0{,}3 \leq \Delta_3 \leq 0{,}5$ ó $1{,}5 \leq \Delta_3 \leq 1{,}7$. Este conjunto evalúa la robustez del diagnóstico para desviaciones de los parámetros que son exclusivamente mayores o menores a los límites especificados para el entrenamiento.

La Figura 3.12 presenta los resultados de la clasificación realizada por los tres clasificadores de cada tipo, con respecto a los conjuntos de prueba considerados.

3.5.2.3. Conclusión de la validación

De los resultados presentados en las Figuras 3.11 y 3.12 se puede concluir lo siguiente:

1. los clasificadores J48 y MLP asociados al caso únicamente de detección (i.e. 2 clases), pueden detectar de manera confiable (i.e. $> 95\,\%$ de clasificaciones son correctas) condiciones anormales en el comportamiento del proceso del caso de estudio. Aún en presencia de incertidumbres no consideradas (conjunto 2), se mantiene una tasa mayor a 92 % de clasificaciones correctas. Los dos tipos de clasificadores pueden identificar de manera confiable fallas de un tamaño mayor al considerado por el diseño;

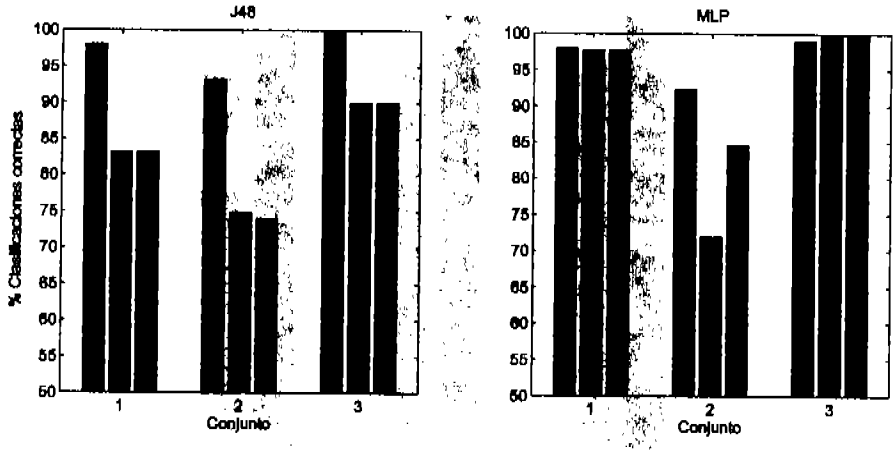


Figura 3.12: Comparación de resultados de la evaluación de robustez del diagnóstico

2. los clasificadores J48 y MLP asociados al caso de detección y aislamiento del parámetro que se desvió (i.e. 3 clases), pueden detectar y aislar de manera confiable, si no hay presencia de incertidumbres no consideradas. En presencia de incertidumbres, la tasa de clasificaciones correctas puede bajar hasta 72 %. Para fallas de mayor tamaño que lo considerado por el diseño, el MLP arroja una tasa mayor de clasificaciones correctas que el J48; y

3. los clasificadores J48 y MLP asociados al caso de detección y aislamiento del parámetro que desvió y la dirección de la desviación (i.e. 5 clases), se pueden detectar y aislar de manera confiable, si no hay presencia de incertidumbres no consideradas. En presencia de incertidumbres no consideradas, la tasa de clasificación es un poco mayor en el caso del clasificador MLP. Para fallas de mayor tamaño que lo considerado por el diseño, el MLP arroja una tasa mayor de clasificaciones correctas que el J48.

Estos resultados validan la propuesta de solución, dado que resuelven el problema diagnóstico definido en la Sección 3.3 y proporcionan un diagnóstico confiable y relativamente robusto. Para el caso de estudio se propone el uso del clasificador de caja-negra, tipo MLP, el cual facilita un diagnóstico relativamente más confiable que el J48.

Finalmente se hace notar, que existe la posibilidad de aplicar varios clasificadores al mismo tiempo y, consecuentemente, aunque no se puede obtener un diagnóstico en todos los casos, es posible detectar casi todas las condiciones definidas como comportamientos anormales. En el caso de estudio de este trabajo de investigación, la limitación en el aislamiento proviene del hecho que solamente una señal esta disponible y es utilizada. Esta limitación es conocida en métodos analíticos, donde, de acuerdo con condiciones de aislamiento para un modelo lineal genérico con dos sensores, solo dos fallas se pueden aislar con un generador de residuos (White y Speyer, 1987).

Capítulo 4

Infraestructura para un sistema automático de operación

4.1. Introducción

Hoy en día los procesos automatizados se utilizan en muchas industrias. Estos procesos cuentan con un sistema automático de operación (SAOP), el cual usualmente proporciona:

1. mecanismos automáticos para lograr la eficiencia del proceso durante su comportamiento normal; y

2. mecanismos para el monitoreo y la protección de peligro en caso de comportamientos anormales; tanto automáticos, como por medio de un operador humano.

La parte de los mecanismos dedicados al comportamiento anormal, se denominan mecanismos de supervisión (Isermann, 1997). En la presentación y descripción de Isermann (1997), la supervisión consta de varios niveles, mostrados en la Figura 1.1, de los cuales hasta ahora sólo el nivel de supervisión con monitoreo y protección se ha integrado como estándar en implementaciones de sistemas automáticos de operación. Este no es el caso del nivel de supervisión con diagnóstico de fallas.

El hecho de que el nivel de supervisión con diagnóstico de fallas no se ha integrado como estándar todavía se debe a:

- problemas abiertos en el área del diagnóstico de fallas, los cuales se presentan y discuten en el Capítulo 3; y

- la complejidad de la tarea de la implementación de un SAOP.

Esta complejidad proviene en principio de las siguiente fuentes:

1. el aumento en la funcionalidad que se requiere del sistema para la tolerancia a fallas, lo cual se puede observar en el número de niveles de supervisión y en sus funciones correspondientes, indicados en el esquema de la Figura 1.1; y

2. el aumento en la necesidad de interacción entre los diferentes niveles y funciones del SAOP, que acompaña a la realización de niveles de supervisión (i.e. monitoreo, protección y supervisión con diagnóstico de fallas); esto se puede observar en el número de conexiones indicadas en la Figura 1.1.

Además, por razones económicas, y por un aumento constante en la complejidad de los mismos procesos industriales, la industria debe tomar en cuenta los siguientes dos escenarios:

1. la extensión o adaptación de una planta existente a lo largo de su ciclo de vida; y
2. la adaptación a diferentes diseños de plantas con diferente estructura e instrumentación, incluyendo sensores y actuadores.

En estos dos escenarios se requiere una solución eficiente para:

- mejorar la confiabilidad en la operación o el rendimiento de un proceso existente; o
- la implementación de un proceso nuevo.

donde "solución eficiente" no solamente implica lograr el efecto esperado, si no también, minimizar la inversión de recursos y esfuerzo para la realización o adaptación del sistema automático de operación.

Para resolver esta problemática, Heck et al. (2003) y Halang et al. (2006) sugieren la creación de sistemas modulares, y se refieren tanto a las partes de hardware, como a las de software. Sin embargo, la modularización tiene como consecuencia factores que aumentan aún más la complejidad de la tarea de implementación de un SAOP, por:

1. la necesidad de comunicación que exige una arquitectura modular; y
2. el esfuerzo de la integración de módulos de hardware y software para formar un sistema integrado, particularmente cuando se trata de módulos heterogéneos en su origen.

La Figura 4.1 presenta el SAOP como sistema modular, indicando los niveles de supervisión definido por Isermann (1997).

Por otra parte, dado el hecho que hoy en día la mayoría de los sistemas son principalmente digitales, se ha observado una disminución en la parte del hardware, simultáneamente con un aumento en la parte del software, que juntos integran un SAOP. De acuerdo con Heck et al. (2003) y Halang et al. (2006) es la parte menor, aproximadamente del 20 % al 40 %, del esfuerzo invertido en el desarrollo y la implementación de un SAOP, que se invierte en la parte de hardware y en la realización de una interfaz entre la parte de software del SAOP y la planta. Está interfaz es comúnmente llamada "nivel de campo", y consiste en:

1. al menos un módulo de software, llamado interfaz-máquina-máquina (IMM), el cual puede mediar entre otros módulos de software del SAOP y módulos de hardware que tienen enlaces con el nivel de campo;
2. al menos un módulo de hardware, que tiene un sistema de entradas y salidas eléctricas, que cuenta con un mecanismo para la conversión entre señales eléctricas y digitales (i.e. convertidores A/D);

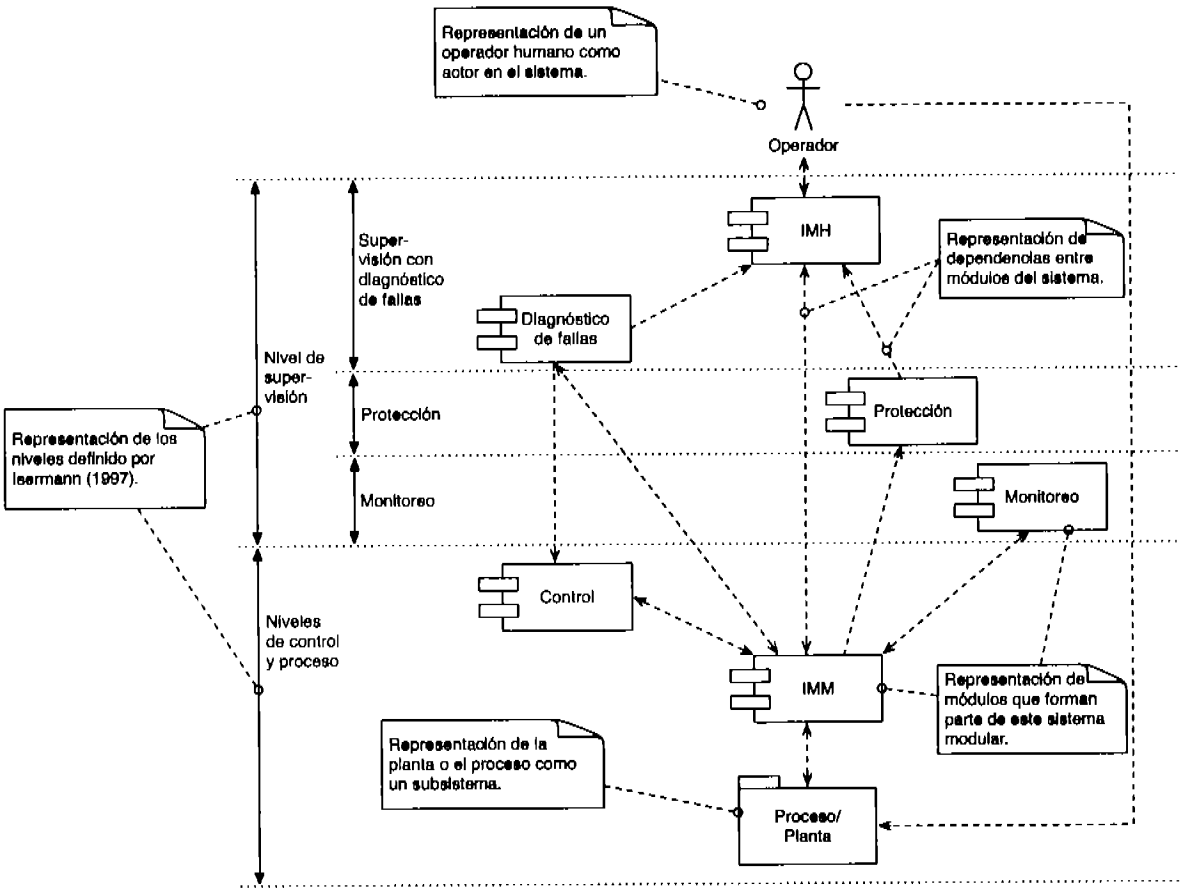


Figura 4.1: Sistema automático de operación (SAOP) representado por medio de un diagrama que utiliza notación definido por UML de manera genérica. En la parte de la izquierda se indica la relación con el esquema general de funciones y niveles de supervisión presentado en la Figura 1.1.

3. varios sensores y actuadores; y

4. los enlaces correspondientes.

El esquema de un nivel de campo se presenta en la parte superior de la Figura 4.2 (izq.), junto con un ejemplo de realización industrial con equipo de la compañía Beckhoff.

Como consecuencia de la disminución en la parte del hardware, la mayor parte del esfuerzo necesario para el desarrollo y la implementación de un SAOP, aproximadamente del 60 % al 80 %, consiste en el desarrollo y la implementación de software y mecanismos de comunicación. Halang et al. (2006) describen que está evolución en los SAOPs ha iniciado un acercamiento entre las áreas de control y de tecnologías de información y comunicación; sin embargo, los autores lo presentan como un reto, ya que la realización:

1. es una tarea compleja; y

2. requiere de conocimientos que típicamente no son parte del perfil de un ingeniero de control (Halang et al., 2006; Heck et al., 2003; Sanz et al., 2000).

Finalmente, se hace notar que las consideraciones anteriores afectan tanto a sistemas de gran escala, como a los de pequeña escala, especialmente cuando:

- el costo de inversión en el diseño y la implementación del SAOP es un factor importante, como en el caso de estudio; y

- se trata de un sistema en desarrollo, que requiere de adaptaciones conforme va evolucionando la validación del proceso.

Estos hechos, junto con el reto antes mencionado, motivaron a:

1. investigar y seleccionar una arquitectura básica de software que permita y promocione la modularidad; e

2. investigar y desarrollar mecanismos de comunicación, los cuales permitan la formación de un sistema distribuido que facilite la integración de:

   a) módulos de software diseñados basados en la arquitectura básica; y

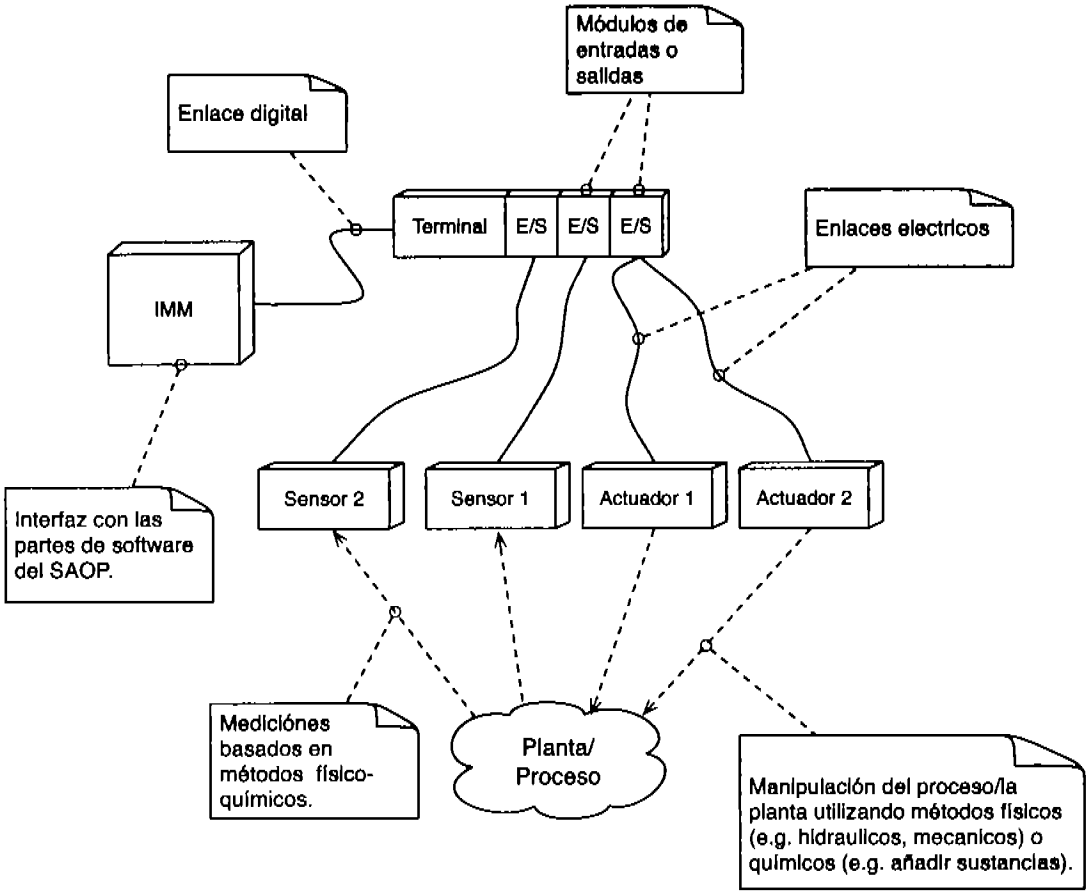
   b) módulos de software basados en otras tecnologías y arquitecturas.

La realización de la arquitectura básica de software y los mecanismos de comunicación, que se presentan en las siguientes secciones, forman una infraestructura para el diseño de un SAOP, que:

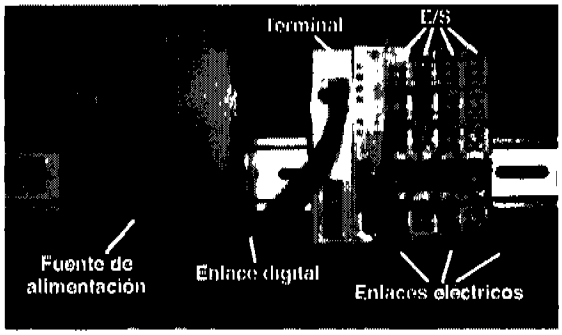
- proporciona el fundamento para la modularidad;

- es abierto y por lo tanto facilita la integración de módulos basados en otras tecnologías y arquitecturas (i.e. la base de la heterogeneidad); y

- permite el manejo de aplicaciones dentro del marco de referencia de las características generales del caso de estudio, presentadas en el Capítulo 2 y resumidas en la Sección 2.3.



(a)



(b)

Figura 4.2: Esquema del nivel de campo, que conecta el SAOP con el proceso (arriba) y ejemplo de una realización industrial (abajo).

Se hace notar que la presente tesis utiliza el vocabulario y los conceptos básicos del diseño y la programación orientado a objetos (DOO y POO). Este vocabulario y los conceptos se introducen en el Apéndice B. Además, para la presentación visual, se utilizan los diagramas y especificaciones del lenguaje unificado de modelado UML, por sus siglas en ingles: unified modeling language. El UML es un lenguaje estandarizado (Object Management Group, 2006b) para el modelado de sistemas de software orientado a objetos, tal como de otros tipos de sistemas, dominios y procesos, el cual permite visualizar, especificar, construir y documentar el sistema. Este lenguaje se introduce junto con guías de notación en el Apéndice A.

4.2. Formulación del problema de la infraestructura de un SAOP

4.2.1. Arquitectura básica de software

En el contexto del área de control y automatización de sistemas complejos, Sanz et al. (2000) describen un marco de referencia para una arquitectura de software que permita manejar adecuadamente tanto la modularidad como la extensión de un sistema automático de operación durante su ciclo de vida. Este marco de referencia para la arquitectura incluye:

1. un lenguaje para la definición de interfaces de componentes;
2. un formato binario para los componentes y un mecanismo que permita cargarlos a la memoria y ejecutarlos durante el tiempo de corrida;
3. una convención para utilizar métodos de un objeto o interfaz;
4. mecanismos para:
   a) manejar y utilizar tipos polimorfos;
   b) recuperar memoria liberada;
   c) encontrar instancias de componentes e identificar sus propiedades;
   d) crear instancias de los componentes durante el tiempo de corrida; y
   e) manejar diferentes versiones de componentes y sus instancias.

El término "durante el tiempo de corrida" se refiere al hecho que no se requiere la terminación y reinicio del proceso o la aplicación correspondiente.

Aparte de las propiedades requeridas de la arquitectura para la parte del software de un SAOP, el proceso de su diseño abarca varias disciplinas. Es necesario integrar los puntos de vista relacionado a:

1. el área de control, donde el enfoque principal son los métodos y algoritmos;
2. el área de ingeniería de procesos, donde el enfoque principal es la funcionalidad del sistema total; y
3. el área de tecnologías de información y de computación, donde el enfoque principal es el diseño y la implementación del SAOP y la integración de las partes de hardware y del software del sistema.

El proceso debe tener como objetivo la creación de sistemas modulares y distribuidos, dentro del marco de referencia de la arquitectura básica de software, mediante el ensamblaje de módulos de software que son producto de subprocesos de:

- el modelado de campo de la aplicación, como por ejemplo, la planta y el proceso del caso de estudio descrito en el Capítulo 2;
- el desarrollo de métodos y algoritmos, como por ejemplo, el diagnóstico descrito en el Capítulo 3; y
- la implementación de una comunicación entre módulos.

La Figura 4.3 presenta un resumen visual de este proceso.

Hoy en día, las arquitecturas modulares de software están frecuentemente basadas en tecnologías orientadas a objetos, las cuales proporcionan:

1. la base para la modularidad, por medio del concepto de objetos;
2. la posibilidad de reuso:
   a) al nivel de clases;
   b) por medio de diseños, los cuales utilizan interfaces y patrones de diseño de Gamma et al. (1995); y
   c) al nivel de paquetes y módulos ensamblados de clases e interfaces (Heineman y Councill, 2001).

En el contexto de diseño orientado a objetos (DOO), los módulos de software son componentes informáticos, los cuáles consisten en:

1. el código ejecutable de clases e interfaces; y
2. datos guardados en archivos que representan recursos necesarios para la ejecución del código binario (e.g. de configuración).

Los componentes, como en el caso de las clases, pueden tener instancias al tiempo de corrida de la aplicación o del proceso. Éstos consisten principalmente en un conjunto de objetos, que son instancias de las clases del componente y se iniciaron con un estado que refleja usualmente la configuración que forma parte del componente.

Aunque se han desarrollado tecnologías de software que proporcionan arquitecturas basadas en componentes en el área de desarrollo de software y en el de tecnologías de información:

- el uso de éstas arquitecturas no es la práctica actual en el diseño de SAOP's (Halang et al., 2006; Heck et al., 2003; Sanz et al., 2000); y
- usualmente no cumplen con todo el marco de referencia descrito por Sanz et al. (2000).

Por lo tanto, se busca una solución que consiste en la selección y adaptación de una arquitectura básica de software, la cual permita y facilita:

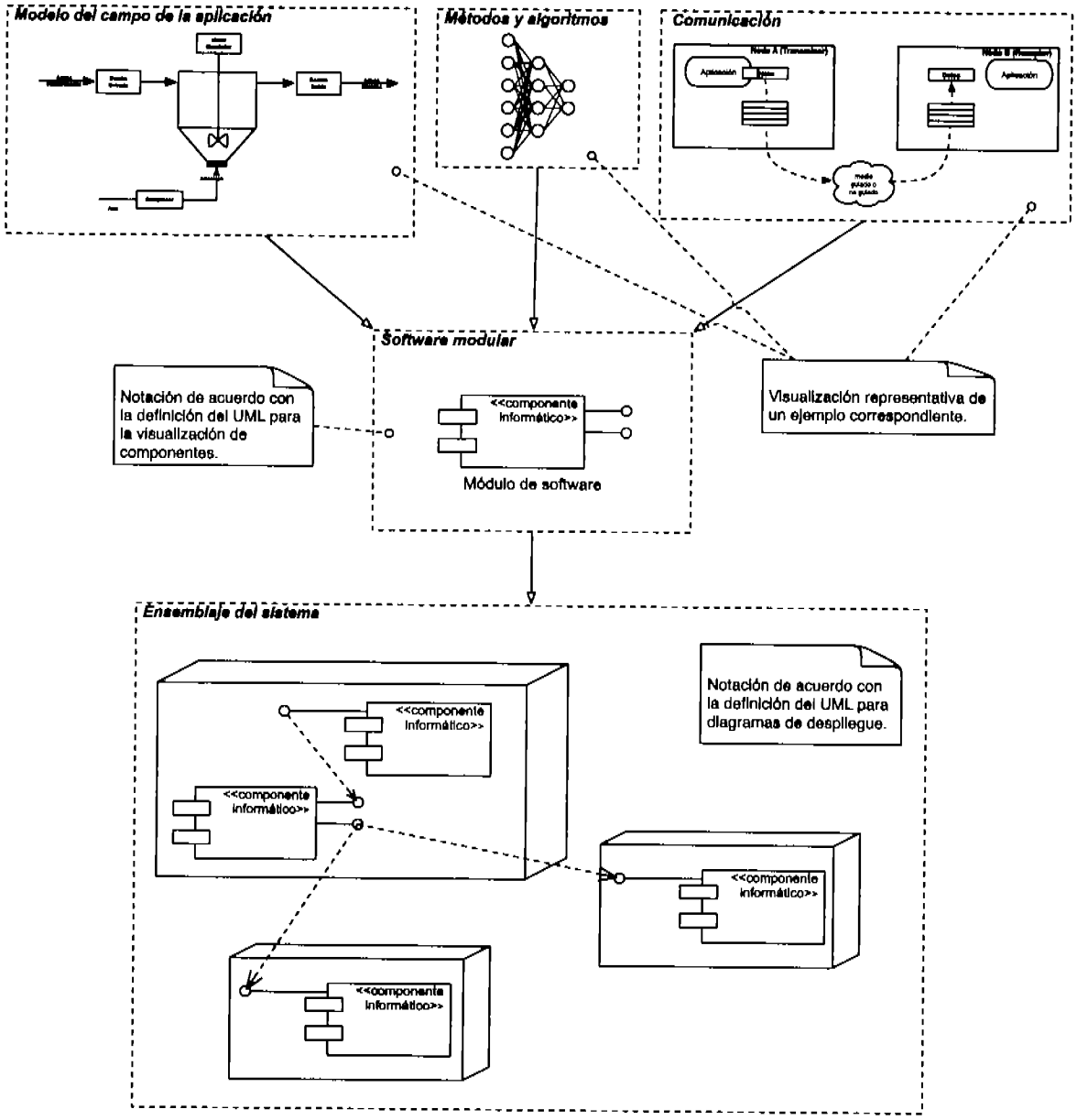


Figura 4.3: Resumen del proceso de desarrollo de un SAOP con una arquitectura modular y distribuida.

1. el análisis orientado a objetos;
2. el diseño y la implementación orientado a objetos;
3. la creación de módulos de software en forma de componentes, descritos anteriormente; y
4. un proceso de diseño y desarrollo que pueda integrar todos los puntos de vista mencionados anteriormente.

4.2.2. Mecanismo de comunicación

Para el despliegue y la interacción de componentes informáticos en un SAOP distribuido que cuenta con varios nodos de hardware, se requiere un mecanismo de comunicación, el cual debe encargarse del envío de datos de un componente en un nodo A (el transmisor) a otro componente en un nodo B (el receptor). En un sistema abierto que permita la heterogeneidad de los nodos que lo integran, la realización del mecanismo de comunicación tiene varios aspectos que pueden ser modelados e implementados con base en el modelo de ISO/OSI (Zimmermann, 1980; Apéndice ??C).

En el marco de referencia de una arquitectura de software para un SAOP distribuido, Heck et al. (2003) presentan los siguientes dos puntos como retos:

1. la complejidad de la programación al nivel de una red; y
2. el hecho de que los conocimientos requeridos típicamente no son parte del perfil de un ingeniero de control.

Tanto Heck et al. (2003) como Sanz et al. (2000), proponen el uso de una abstracción en forma de un subsistema de comunicación, el cual maneja los detalles del intercambio de datos entre componentes, de manera que no importe el despliegue físico de los componentes (i.e. locales o remotos). Este subsistema puede consistir en uno o varios componentes, debe presentar una interfaz sencilla al programador que realiza un componente informático para un SAOP.

Entonces, se busca modelar e implementar una pila de capas para proporcionar el mecanismo de comunicación, el cuál debe contar con lo necesario para satisfacer las necesidades de un SAOP, particularmente con el tamaño y con las características parecidas al caso de estudio de la presente tesis (Sección 2.3):

1. la capa de aplicación debe proporcionar un servicio con una interfaz sencilla a los otros componentes del SAOP;
2. el transporte de paquetes de datos debe cumplir con límites temporales;
3. la red utilizada debe ser de bajo costo; y
4. la implementación debe contar con un mecanismo que facilite la configuración.

Aspectos temporales de la comunicación

Neumann (2006) propone una clasificación de requisitos para la comunicación en tiempo real, RT (por sus siglas en inglés *real-time*), en el área de automatización, la cuál tiene como enfoque el comportamiento del tiempo de respuesta de paquetes de datos. Esta clasificación consta de las clases mostradas en la Tabla 4.1.

Clase	Nombre	Tiempo de ciclo
1	RT suave	ajustable
2	RT duro	1-10 ms
3	RT isócrono	250μs-1ms

Tabla 4.1: Clasificaciones de tiempo real definidas en términos del tiempo de ciclo en el transporte de paquetes de datos presentado por Neumann (2006).

Para la realización de un SAOP para un proceso en la escala del caso de estudio, se busca el transporte de paquetes de datos con la definición del tiempo real suave (clase 1), considerando un tiempo de ciclo menor a 1 segundo.

Mecanismo de configuración

Finalmente, se requiere de un mecanismo de auto-configuración para el sistema distribuido, el cual garantiza que no se necesitan conocimientos explícitos de direcciones o números de puertas para establecer conexiones entre nodos, respectivamente entre los componentes informáticos colocados en ellos.

4.3. Solución al problema de infraestructura de un SAOP

4.3.1. Arquitectura básica de software

Aunque existen muchos lenguajes de programación orientados a objetos, se propone el uso de:

1. el lenguaje y la plataforma Java (Lindholm y Yellin, 1999; Gosling et al., 2005); y
2. OSGi (OSGi Alliance, 2006), una especificación para Java que estandariza el desarrollo de componentes informáticos;

ya que estas dos tecnologías juntas, permiten la creación de sistemas modulares en el marco de referencia definido por Sanz et al. (2000) para una arquitectura orientada a objetos y componentes informáticos para un SAOP.

Java

El lenguaje Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 1990. Programas desarrollados en Java típicamente se compilan a un código ejecutable por una máquina Java, la cuál puede interpretar y compilar el código para la ejecución. Desde la implementación original y de referencia publicada alrededor de 1995 por Sun Microsystems, Java consiste en:

- una máquina de referencia virtual;
- el lenguaje de programación; y

- una biblioteca de clases que proporciona funcionalidad básica para aplicaciones, como es el manejo de datos.

El término "virtual" se refiere al hecho de que se trata de una emulación, la cual se ejecuta en una máquina real.
En conjunto estos elementos que forman parte de la plataforma Java facilitan:

1. un lenguaje para la definición de interfaces de componentes;
2. un formato binario para los componentes y un mecanismo que permita:
   a) que se pueda cargar a la memoria; y
   b) ejecutar al tiempo de corrida;
3. una convención para utilizar métodos de un objeto o interfaz;
4. mecanismos para:
   a) crear instancias de los componentes al tiempo de corrida;
   b) manejar y utilizar tipos polimorfos; y
   c) recuperar memoria liberada.

OSGi

OSGi (por sus siglas en inglés: Open Service Gateway interface) es una especificación estandarizada desde el año 2000, la cual está basada en la plataforma Java, para una arquitectura que puede manejar componentes y servicios de manera dinámica, originalmente diseñada para sistemas limitados en recursos (en inglés *embedded systems*).
Esta especificación y su implementación, en forma de un contenedor para los componentes, proporciona mecanismos para:

1. encontrar componentes y sus propiedades;
2. manejar versiones de componentes;
3. declarar y especificar componentes con respecto a su interfaz;
4. desplegar componentes al tiempo de corrida (añadir, remover, actualizar);

Además, proporciona especificaciones de servicios básicos, e implementaciones de referencia disponibles también en forma de componentes, como por ejemplo, para el manejo de una bitácora o de la configuración de otros componentes. Éstos facilitan el desarrollo de componentes, ya que permiten al programador concentrarse en la aplicación.
El componente informático en el contexto de OSGi es denominado "bulto" y consiste en:

- clases e interfaces:
  1. públicas (disponibles para el uso por medio de otros componentes);

2. privadas (no disponibles para el uso por medio de otros componentes); y

- archivos de configuración; mínimo un archivo descriptor del bulto que permita a los mecanismos del contenedor OSGi manejar al componente.

El término "archivo descriptor" se refiere a un archivo con datos de configuración de un bulto, el cuál contiene información acerca del bulto, como por ejemplo:

- el nombre;
- una descripción;
- un identificador único;
- la versión;
- el ambiente que es necesario para la ejecución;
- etiquetas que especifican categorías;
- el activador;
- las dependencias que requiere, en forma de paquetes de Java, los cuales se denominan "exportaciones"; y
- los servicios y el modelo que proporcionan a otros componentes en forma de paquetes de Java, los cuales se denominan "importaciones".

El "activador" es la implementación de una interfaz llamada `Activator` que forma parte del modelo especificado por OSGi, y las "exportaciones" e "importaciones" permiten al contenedor resolver las dependencias al momento de instalar un bulto. En la presente tesis todos los bultos son componentes informáticos con la estructura de paquetes que se muestra en la Figura 4.4.

El contenedor OSGi maneja bultos de acuerdo con la especificación del ciclo de vida de bultos en OSGi, el cual se presenta en la Figura 4.5. Este ciclo cuenta con los siguientes estados:

- *Instalado*: el cual indica que un bulto ha sido instalado dentro del contenedor;
- *Resuelto*: el cual indica que un algoritmo para la verificación de restricciones ha terminado exitosamente, y consecuentemente, que todas las partes públicas de otros bultos de las cuales depende el bulto, están disponibles.
- *Arrancando*: el cual indica que el contenedor OSGi está iniciando la implementación del bulto, por medio de un mecanismo que se denomina activador;
- *Activo*: el cual indica que el contenedor OSGi ha iniciado exitosamente el bulto, y que las partes públicas del bulto están disponibles para el uso por medio de otros bultos; y
- *Deteniendo*: el cual indica que el contenedor OSGi está desactivando la implementación de un bulto, también por medio del activador, el cual se utiliza para el arranque;

Esta parte de la especificación de OSGi facilita la dinámica del sistema, ya que no se requiere la reinicialización del contenedor para añadir, remover o actualizar bultos.

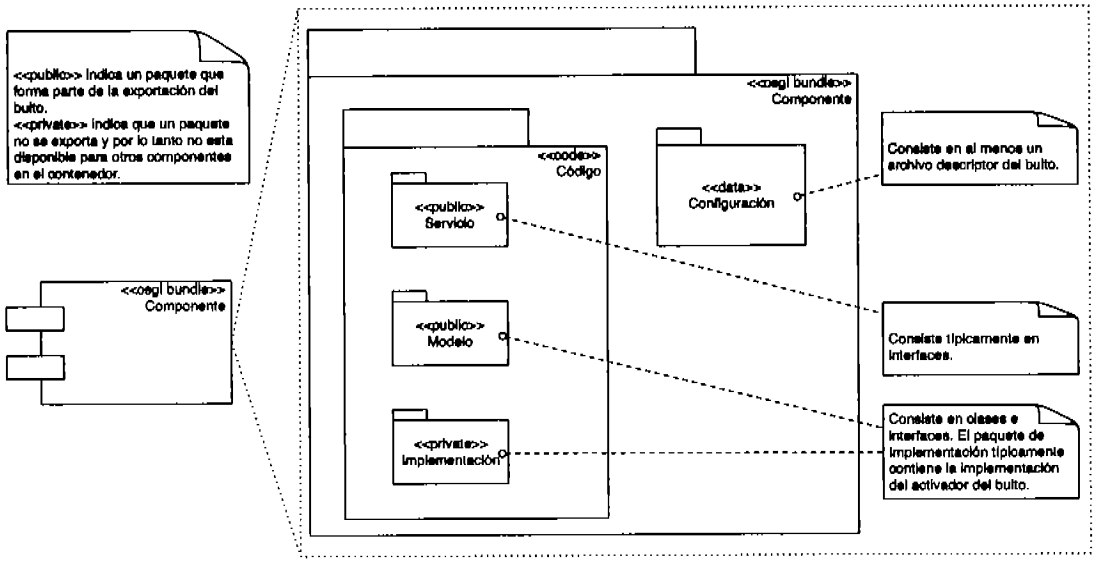


Figura 4.4: Estructura de un bulto OSGi de acuerdo con la definición del UML de la notación para diagramas de componentes (izquierda) y diagramas de paquetes (derecha). En la implementación los paquetes siempre corresponden a paquetes como lo define el lenguaje Java.

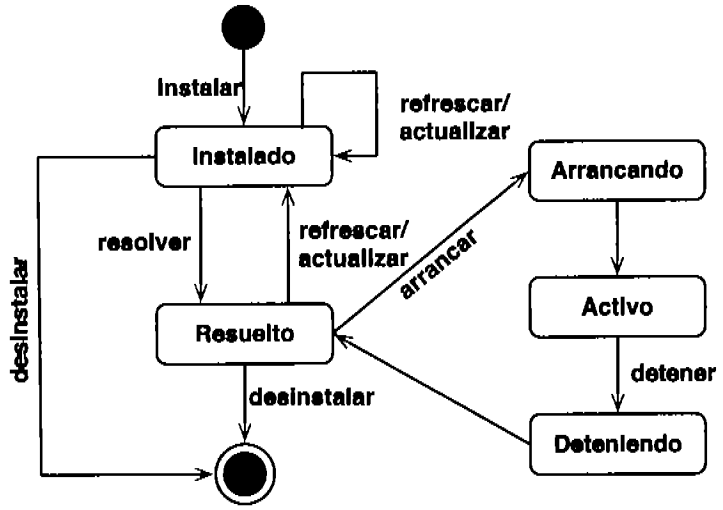


Figura 4.5: Estados y transiciones en el ciclo de vida de un bulto OSGi, de acuerdo con la definición del UML de la notación para diagramas de estados.

Diseño y despliegue de bultos

La arquitectura básica de software basada en la plataforma Java y la especificación OSGi, permite y facilita:

1. el análisis orientado a objetos;
2. el diseño y la implementación orientado a objetos;
3. la creación de módulos de software en forma de componentes informáticos denominados bultos; y
4. un proceso de diseño y desarrollo que pueda integrar los puntos de vista mencionados en la Sección 4.2.1 de la formulación del problema de la arquitectura básica de software.

La Figura 4.6 muestra un resumen del desarrollo y despliegue de un bulto OSGi.

La parte más importante para el diseño es la parte pública de un componente, la cual se exporta para ser utilizada por otros componentes del sistema. La parte de la implementación que no se exporta, puede ser reemplazada por otras implementaciones, hecho que proporciona parte de la flexibilidad de esta arquitectura.

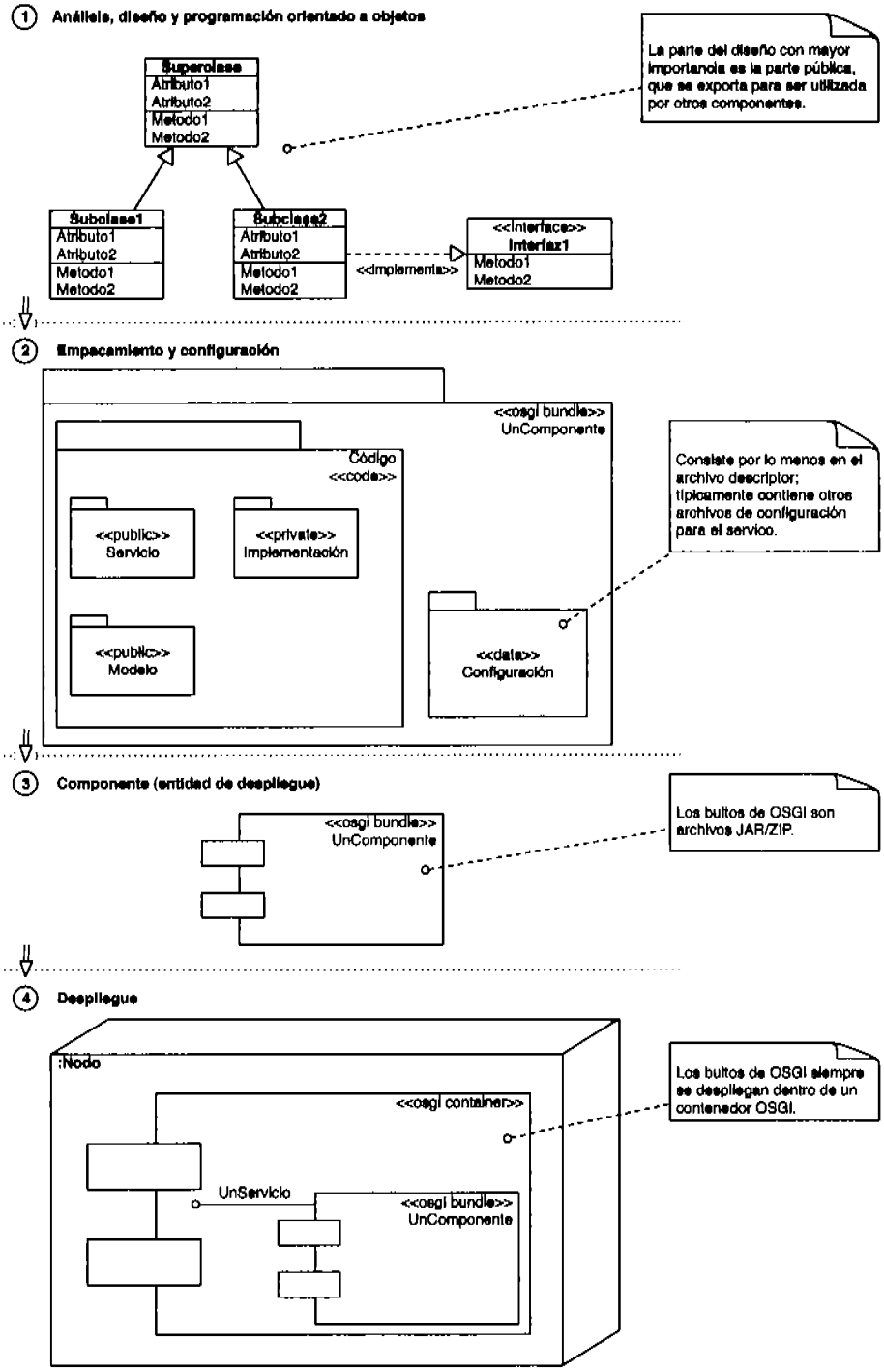


Figura 4.6: Desarrollo y despliegue de un bulto OSGi

4.3.2. Mecanismo de comunicación

El mecanismo de comunicación se implementó con varios protocolos, basado en el modelo de referencia ISO/OSI. La especificación del mecanismo de comunicación por medio de protocolos tiene la ventaja de permitir la integración de componentes heterogeneos en un SAOP, los cuales se desarrollaron fuera de las especificaciones y el marco de referencia de la arquitectura definido en la presente tesis.

La pila del presente trabajo tiene dos atributos principales:

1. un protocolo para la capa de aplicación, que permita proporcionar un servicio con una interfaz sencilla a los otros componentes de un SAOP; y
2. el transporte de los paquetes de la capa de aplicación se realiza de manera que:
   a) pueda cumplir con límites temporales para la transmisión; y
   b) que la red utilizada sea de bajo costo.

Para el transporte de paquetes a partir de la capa de aplicación, presentado y argumentado con más detalle en la Sección 4.3.2.1, se utilizaron como base:

1. UDP, en la capa de transporte;
2. IP, en la capa de red; y
3. Ethernet para la capa de enlace de datos y la capa física.

El protocolo UDP como parte de la capa de transporte no proporciona:

- el transporte confiable de un paquete (i.e. no cuenta con un mecanismo de detección de errores en la transmisión);
- una secuencia específica de paquetes transportados; y
- un tiempo límite para el transporte.

Consecuentemente, se complementó la capa de transporte con un protocolo de transmisión que aporta estos atributos, bajo la suposición que se busque el transporte de un paquete a todos los nodos participantes. Esta clase de protocolos típicamente se denomina en inglés "reliable ordered multicast" y tiene el acrónimo ROM.

Para la capa de aplicación, presentada y argumentada con más detalle en la Sección 4.3.2.2, se desarrolló un protocolo específico que cumple con el requisito de proporcionar un servicio y, una interfaz sencilla a componentes que forman parte de la aplicación (i.e. el SAOP). Este protocolo modela un espacio de tuplas distribuido sobre los nodos participantes de la red, respectivamente del anillo lógico formado por el servicio de mensajes ROM.

La composición de la pila que finalmente representa el mecanismo de comunicación de la presente tesis, se muestra en la Figura 4.7. Esta figura también muestra la correspondencia de los protocolos a las capas del modelo de referencia ISO/OSI.

Por otra parte, un aspecto que siempre representa un problema para un sistema distribuido es la configuración de los nodos para que pueden participar en la comunicación por medio de una red

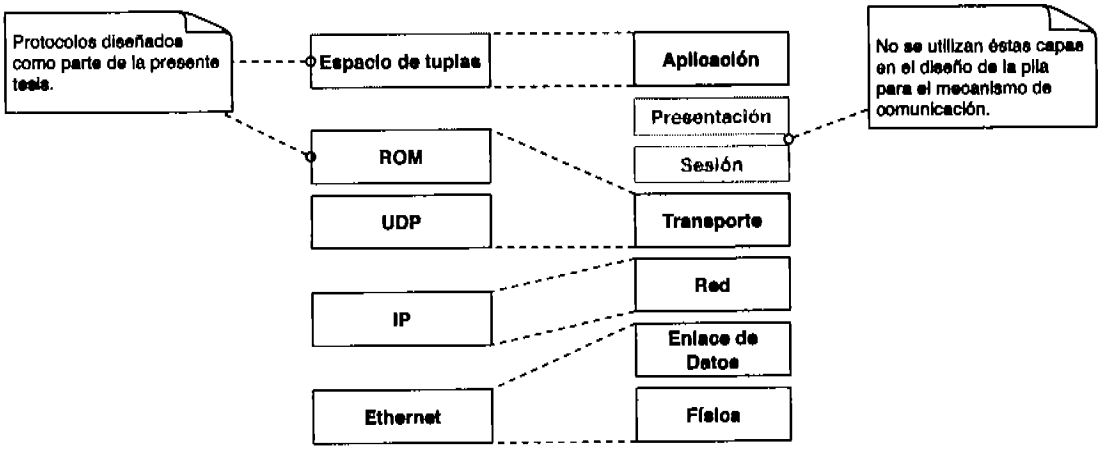


Figura 4.7: Pila de protocolos utilizados para el mecanismo de comunicación, y la correspondencia hacia las capas del modelo de referencia ISO/OSI.

local. Para resolver este problema, se desarrolló un protocolo que permite la auto-configuración, eliminando la necesidad de tener y poder aplicar conocimientos explícitos de direcciones o números de puertas para establecer conexiones entre nodos y sus componentes informáticos. Este mecanismo de auto-configuración de la red basado en un protocolo se presenta y argumenta con más detalle en la Sección 4.3.2.3.

A cada uno de los protocolos desarrollados se aplicó el análisis y diseño orientado a objetos para obtener:

1. una interfaz que presenta un modelo orientado a objetos del servicio que puede prestar el protocolo; y
2. clases e interfaces que presentan un modelo orientado a objetos de los elementos de cada protocolo (i.e. modelo de campo del protocolo).

Estos se utilizan consecuentemente para las partes públicas de la implementación de los componentes correspondientes a cada protocolo. Se hace notar, que el modelo público representa una abstracción, la cuál permite reemplazar la implementación y el protocolo, sin requerir cambios en los otros componentes del sistema.

4.3.2.1. Transporte de paquetes del nivel de la aplicación

Existen muchos tipos de productos y tecnologías para implementar redes de control (Neumann, 2006; Schickhuber y McCarthy, 1997). No obstante, en su mayoría son productos integrados de compañías o alianzas de compañías, las cuales:

- típicamente requieren de una inversión inicial muy grande;
- no están fácilmente disponibles en cualquier parte del mundo; y
- no son plataformas abiertas que permiten la heterogeneidad.

La tecnología de Ethernet representa una alternativa que es:

1. de fácil disponibilidad en el mercado;
2. de bajo costo; y
3. una plataforma abierta.

Estas son las mismas razones, por las cuales hoy en día hay mucho interés en el uso de Ethernet para SAOP's en el área de la automatización industrial (Neumann, 2006).

Neumann (2006), así como Jasperneite y Watson (2003), afirman que la utilización de Ethernet para SAOPs es posible en casi todas las clases definidas por la Tabla 4.1; sin embargo, mencionan que para el uso en tiempo real duro o isócrono se requiere una optimización de la comunicación en las capa de enlace de datos y la capa física proporcionada por Ethernet. Esta optimización únicamente se puede realizar con hardware y equipo especialmente diseñado, lo cual, como derivación de la tecnología Ethernet, puede tener un bajo costo en comparación a otros productos comparables.

Para el caso de RT suave se puede utilizar equipo de Ethernet estándar. Sin embargo, el reto para realizar una comunicación en tiempo real es el modo de operación estándar del Ethernet CSMA/CD (de sus siglas en inglés: Carrier Sense Multiple Access/Collision Detection). En particular, la resolución de colisiones no es determinista en el tiempo, dado que cuando ocurre una colisión el nodo espera un tiempo aleatorio hasta intentar una retransmisión.

Este problema se puede resolver utilizando un mecanismo en la capa de transporte que pueda asegurar el acceso exclusivo al nivel físico, evitando el problema de las colisiones. En la presente tesis se propone para este mecanismo la utilización de un protocolo de transmisión ROM, en el cual el grupo de procesadores (nodos del sistema) forman un anillo lógico y pasan un testigo que asegura el acceso exclusivo al nivel físico de la red. Este protocolo se describe con más detalle en la Sección 4.3.3.

La pila ensamblada de protocolos encargados del transporte se presenta en la Figura 4.8, junto con el ensamblaje de paquetes de datos mediante el mecanismo de encapsulamiento, pasando por todas las capas participantes.

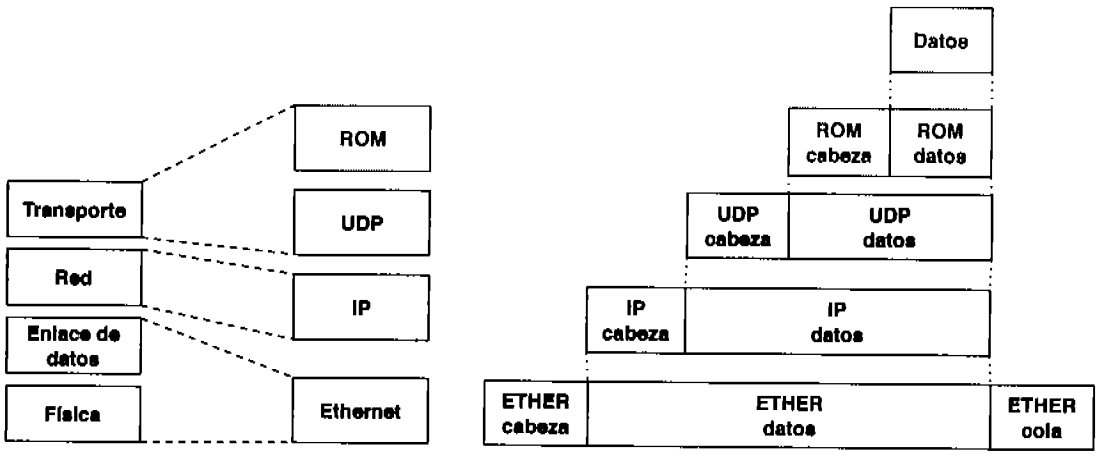


Figura 4.8: Pila de protocolos hasta la capa de transporte en el modelo ISO/OSI (izq.), y el ensamblaje de paquetes de datos por el mecanismo de encapsulamiento, pasando por todas las capas participantes (der.).

Para optimizar la transmisión y asegurar que funciona el protocolo ROM con las propiedades requeridas, hay que evitar la fragmentación de paquetes en las capas inferiores de la pila, donde típicamente

hay un límite para la cantidad de datos que se pueden transmitir al mismo tiempo. Este límite, la unidad máxima de transporte (MTU), es de 1500 bytes en el caso de Ethernet estándar, del cual se tienen que substraer los tamaños de la información añadida en el proceso de encapsulamiento en forma de cabezas y colas. El MTU de la capa de transporte con aproximadamente 1400 bytes es grande, considerando que:

1. en un SAOP la comunicación típicamente consiste en la transmisión de cantidades pequeñas de datos de manera frecuente (Schickhuber y McCarthy, 1997); y
2. en el caso de estudio, dadas las características descritas en Sección 2.3, hay una cantidad moderada de variables, las cuales tienen que ser intercambiadas entre los nodos que participan en el sistema. e.g. un `double` en una arquitectura de hardware de de 32-bits consiste en 8 bytes, lo cual significa que este MTU permite el transporte de 175 valores del tipo `double`.

4.3.2.2. El nivel de la aplicación

Existe una clase de software denominado *middleware* (Schantz y Schmidt, 2001), la cual maneja los detalles del intercambio de datos entre componentes locales y los que están distribuidos físicamente en diferentes nodos interconectados por una red. Este software típicamente presenta una interfaz sencilla al programador, y reduce la complejidad de la tarea de la implementación de la comunicación en un SAOP, especialmente cuando es un sistema distribuido (Heck et al., 2003).

Típicamente un *middleware* ocupa un papel central en la arquitectura de un sistema distribuido, muy parecido al patrón de diseño de software orientado a objetos denominado *mediador* (Gamma et al., 1995): permite y coordina la interacción de varios componentes sin introducir nuevas dependencias entre ellos. El SAOP presentado en la Figura 4.1 se puede transformar utilizando una arquitectura basado en un *middleware*. Esta nueva arquitectura elimina dependencias entre los componentes, reemplazándolos por la dependencia del mediador (i.e. el *middleware*) como se muestra en la Figura 4.9.

CORBA (Object Management Group, 2006a) es un estándar para llamar procedimientos remotos, que define el intercambio de objetos descritos por GIOP (General Inter-ORB Protocol ó IIOP Internet Inter-ORB Protocol en caso de TCP/IP). Es un *middleware* utilizado en algunos sistemas para el intercambio de objetos, sin embargo:

1. tiene una semántica compleja para la descripción de datos en objetos (GIOP);
2. son las capas a partir de la capa de transporte las que tienen que garantizar un orden y límites en el tiempo de transmisión.

Aunque existe la posibilidad de implementar el nivel del transporte sobre la pila de protocolos descritos anteriormente, la semántica y consecuentemente la interfaz es relativamente complejo para el programador. Por lo tanto se propone como solución una realización basada en el concepto formulado por Gelernter (1992) de un espacio de tuplas (TS, de sus siglas en inglés Tuple Space). Esta solución tiene dos ventajas:

1. tiene una semántica muy sencilla;
2. se puede realizar como un sistema distribuido con un protocolo más sencillo que realizaciones de GIOP.

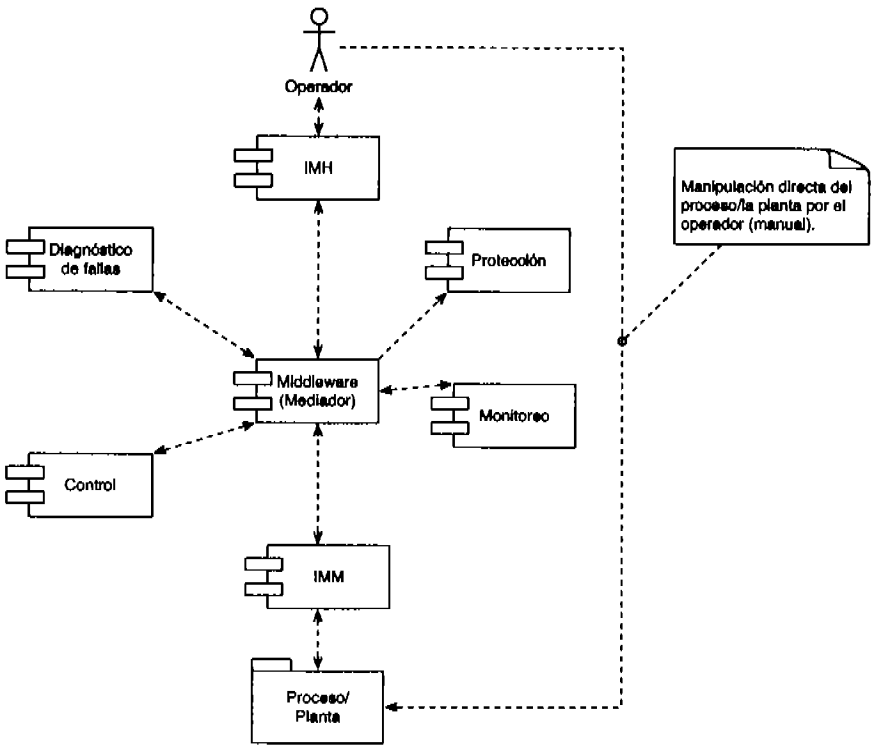


Figura 4.9: Esquema de un SAOP utilizando una arquitectura con un *middleware*.

Bollela et al. (1999) presentaron una propuesta para adaptar una implementación existente a tiempo real, sin embargo:

- es un diseño donde un nodo especial maneja un TS centralizado; y
- no se investigó el problema de la comunicación en tiempo real.

El diseño de un TS distribuido, el cual pueda garantizar tiempo real suave para formar una memoria volátil distribuida, se identificó como un problema abierto, que se investigó en el marco de este trabajo. Las restricciones temporales se pueden garantizar implementando el TS como la capa de aplicación sobre la pila de capas encargadas del transporte, como se muestra en la Figura 4.10, la cual se presentó en la Sección 4.3.2.1.

La definición del TS distribuido y del protocolo que representa la capa de aplicación de la red del SAOP, se describe con más detalle en la Sección 4.3.4. También se presenta la realización del TS como componente de la arquitectura descrita en 4.3.1.

4.3.2.3. Mecanismo de auto-configuración de la red

En un sistema distribuido conectado por una red, usualmente se requieren conocimientos explícitos de direcciones o números de puertas para establecer conexiones entre nodos. En el caso de un SAOP también se requiere esta información como configuración en sus componentes, lo cual muchas veces requiere de la presencia de un experto durante la instalación o la reconfiguración del sistema.

Como alternativa, existe la posibilidad de la auto-configuración del sistema por medio de:

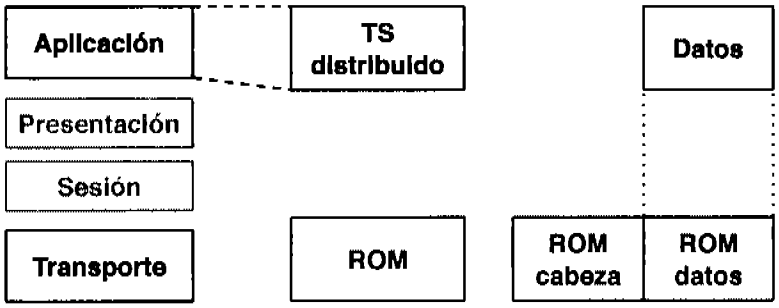


Figura 4.10: Nivel de aplicación sobre la capa de transporte y la pila mostrada en la Figura 4.8.

1. la auto-configuración de la dirección, por medio de auto-asignación de una dirección no ocupada (Steinberg y Chesire, 2005); y
2. la auto-configuración de los conexiones entre componentes por medio del descubrimiento de los servicios participantes (NSD por sus siglas en ingles, Network Service Discovery; Vinkoski (2003); Helal (2002)).

La auto-asignación de una dirección no ocupada en redes IP se ha estandarizado (Cheshire et al., 2005) y ya forma parte de muchos sistemas operativos. Este mecanismo no requiere de un servicio central para asignar o encontrar una dirección no ocupada.

Una vez asignado una dirección IP al nodo, se puede utilizar un protocolo de NSD para encontrar servicios de componentes y establecer conexiones entre ellos. Existen varios protocolos como SLP2 (Guttman, 1999) y mDNS (Steinberg y Chesire, 2005), sin embargo están diseñados para sistemas de mayor escala y cuentan con mecanismos complejos para manejar la dinámica de servicios en estos ambientes. Aparte de la complejidad de estos protocolos, los mecanismos requieren de comunicación iniciada por el agente del protocolo NSD, lo cual significa que no se puede garantizar el acceso exclusivo al medio de comunicación.

Dadas las características del SAOP, se pueden hacer las siguientes suposiciones:

1. la red de área local para un SAOP requiere de un número pequeño de nodos ($<<255$); y
2. el SAOP solo requiere de configuración en el momento de la instalación y cuando hay una ocasional reconfiguración del sistema durante su ciclo de vida.

Bajo estas suposiciones, se propone un protocolo sencillo como solución, el cual puede manejar una red de área local de la escala de un SAOP y asegura que únicamente hay intercambio de mensajes cuando el nivel de aplicación solicita encontrar servicios en la red. La definición de este protocolo se describe con más detalle en la Sección 4.3.5.

4.3.3. Protocolo ROM

4.3.3.1. Modelo básico

El modelo básico del sistema consiste en un conjunto de procesos que interactúan enviando mensajes por medio de un canal de comunicación. Las definiciones de este modelo se presentan en la Tabla 4.2.

m	Mensaje distinguible únicamente
p,q	proceso
Π	Conjunto de todos los procesos $\Pi = \{p_1, p_2, ..., p_n\}$ $n < 255$
Emisor(m)	Proceso $p \in \Pi$, el cual envía un mensaje m
Dest(m)	Conjunto de procesos destinatarios de m ($Dest(m) \subset \Pi$)

Tabla 4.2: Definiciones básicas del modelo

Las propiedades que se buscan en el sistema están definidas con base en la especificación propuesta por Défago et al. (2004):

1. *Validez:* si un proceso correcto manda un mensaje m a *Dest(m)*, entonces el proceso correcto $p \in Dest(m)$ eventualmente entrega m;
2. *Acuerdo Uniforme:* si un proceso entrega un mensaje m, todos los procesos correctos $p \in Dest(m)$ eventualmente entregan m;
3. *Integridad Uniforme:* todos los procesos $p_n \in Dest(m)$ entregan m una sola vez, si y solo sí m ha sido enviado por *Emisor(*m*)*;
4. *Orden Total Uniforme*: si los procesos p y q entregan los mensajes m_1 y m_2, el proceso p entrega m_1 antes que m_2 si y solo sí el proceso q entrega m_1 antes que m_2.

La definición del sistema toma en cuenta la falla total de procesos y fallas en la transmisión de mensajes (e.g. omisión de mensajes). En este contexto, un proceso correcto está definido como un proceso que no presenta ninguna falla.
Para manejar adicionalmente los requisitos temporales, se propone:

1. la formación de un anillo lógico estático por Π cuando se inicia el sistema;
2. el uso de un testigo lógico cuya posesión da al proceso $p_i \in \Pi$ el privilegio de enviar mensajes y de esta manera asegura el acceso exclusivo al medio de comunicación;
3. la circulación del testigo lógico de manera síncrona, adentro de un intervalo de tiempo acotado.

El anillo lógico y la operación básica del anillo se muestran en la Figura 4.11.
El algoritmo básico del protocolo tiene los siguientes pasos:

1. El proceso p que recibe el testigo lógico
   a) si tiene un mensaje:
      - incrementa el contador de mensajes;
      - envía el mensaje directamente al sucesor en el anillo, el cual transmite el mensaje a su sucesor; esto se repite hasta que el mensaje llega otra vez al proceso que posee el testigo lógico;
      - al recibir el propio mensaje otra vez, sigue el paso 2.

   b) si no tiene mensaje, sigue el paso 2.
2. El proceso p que posee el testigo lógico, pasa el testigo con información actualizada al sucesor en el anillo por medio de un multicast.

Se hace notar que el multicast del testigo lógico permite la sincronización de los relojes y asegura que la entrega de los mensajes al nivel de la aplicación ocurre en todos los nodos al mismo tiempo.

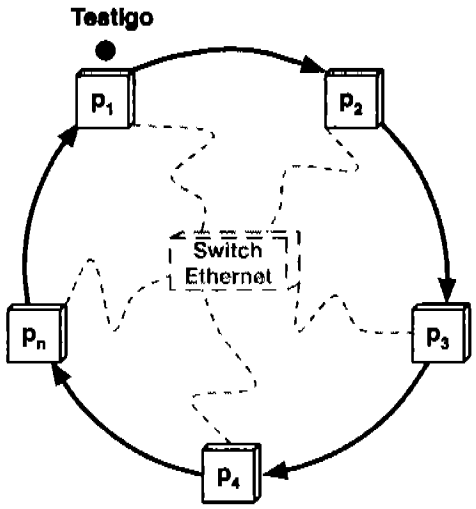


Figura 4.11: Modelo del anillo lógico con testigo lógico

4.3.3.2. Detectores de fallas

Para la detección de las fallas que toma en cuenta la definición del sistema, se propone el uso de dos detectores:

1. un reloj físico y sincronizado en cada procesador $p_i \in \Pi$ que permita detectar la falla total en un proceso $p \in \Pi$;

2. un contador cuyo valor identifica la secuencia de mensajes, el cual se transmite adicionalmente en el testigo lógico para permitir la detección de la omisión de mensajes.

Cada procesador tiene un tiempo máximo en que puede poseer el testigo y utilizar para enviar un mensaje, el cuál se define como t_{rt}. El procesador que posee el testigo tiene que pasarlo lo antes posible, pero definitivamente antes de que expire t_{rt}. Esto garantiza un límite de tiempo de rotación del testigo y permite la detección de

1. la pérdida del testigo si falla el proceso que posee el testigo en este momento;

2. la pérdida de un mensaje en el canal de comunicación.

En el caso que ocurra una falla se puede detectar y manejar al nivel de la aplicación (el SAOP) ya que:

1. la falla total de un proceso tiene un significado para la aplicación (e.g. los componentes del SAOP);

2. la aplicación define las restricciones temporales, dependiendo de las cuales se puede decidir si es posible retransmitir un mensaje perdido en el canal de comunicación.

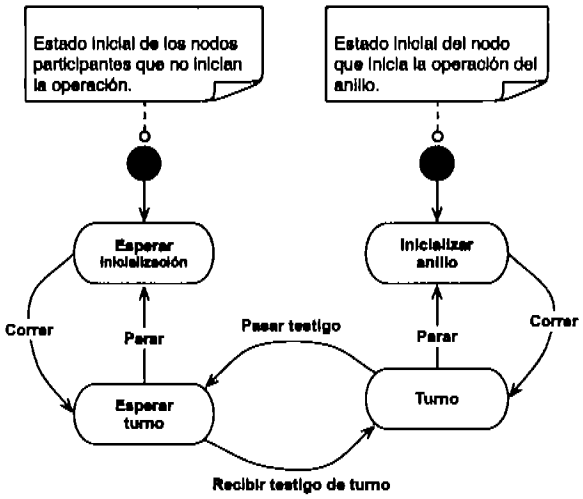


Figura 4.12: Estados y transiciones en la operación del anillo lógico, de acuerdo con la definición del UML de la notación para diagramas de estados.

4.3.3.3. Operación del anillo lógico

Para facilitar la inicialización y la reconfiguración del anillo se propone la operación del anillo por medio de la máquina de estados finitos, mostrada en la Figura 4.12.
La maquina de estados cuenta con los siguientes estados:

- *Estados iniciales*: La configuración del anillo se inicia por un nodo específico, el cual se puede determinar de manera automática (e.g. orden de los identificadores de los procesadores) o de manera manual (e.g. configuración al nivel de la aplicación).
- *Inicializar anillo*, el estado en el cual el nodo inicia al anillo:
  1. usa el servicio del protocolo NSD para detectar todos los nodos participantes;
  2. prepara la lista ordenada de nodos participantes por:
     a) las direcciones IP, las cuales en la versión 4 de IP se pueden expresar como números entero de 4 bytes; ó
     b) prioridades configuradas en las propiedades del servico en cada nodo;
  3. envía la lista a todos los nodos participantes por medio de un multicast.
- *Esperar inicialización*, el estado en el cual todos los otros nodos esperan el mensaje de la configuración del anillo.
- *Turno*, el estado en el cual un nodo posee el testigo lógico y puede enviar un mensaje.
- *Esperar turno*, el estado en el cual un nodo recibe y maneja los mensajes enviados por otros nodos.

Las transiciones entre *turno* y *esperar turno* están determinadas por el algoritmo básico del protocolo. Las otras dos posibles transiciones permiten la inicialización del anillo y la reconfiguración cuando se considera necesario:

- *Correr*:
  El nodo que inicia el sistema crea el testigo lógico y empieza a enviar un mensaje o a pasar el testigo lógico. Todos los nodos operan de acuerdo con el algoritmo básico del protocolo descrito anteriormente.

- *Parar*:
  El nodo que inició el sistema espera su turno y cuando recibe el testigo lógico detiene la circulación, avisando a los nodos participantes por medio de un multicast.

4.3.3.4. Modelo público

El modelo público diseñado para el mensajero ROM se muestra en el diagrama de paquetes en la Figura 4.13 y el diagrama de clases en la Figura 4.14. Para el transporte de paquetes de datos al nivel de la aplicación se requiere la realización del modelo, en particular, la implementación de las interfaces `MessageData` y `MessageDataFactory`.

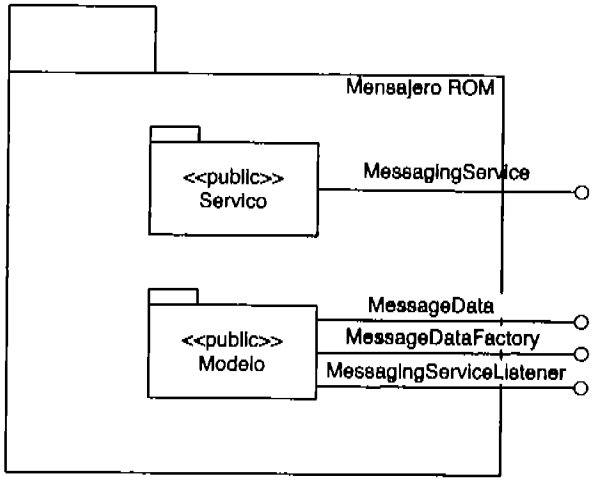


Figura 4.13: Diagrama de paquetes del modelo público del mensajero ROM, de acuerdo con la definición del UML.

El `MessageDataFactory` es un patrón de creación (Gamma et al., 1995), el cual permite a la implementación del mensajero ROM crear los paquetes de datos para el nivel de la aplicación sin conocimientos detallados del contenido de los paquetes.

4.3.3.5. Paquetes del protocolo

Los paquetes del protocolo están compuestos de los siguientes elementos:

- *tipo de mensaje*, un byte que define el tipo del mensaje ROM, el cual puede indicar:
  - un mensaje de datos;
  - el testigo lógico;
  - un mensaje de inicialización.

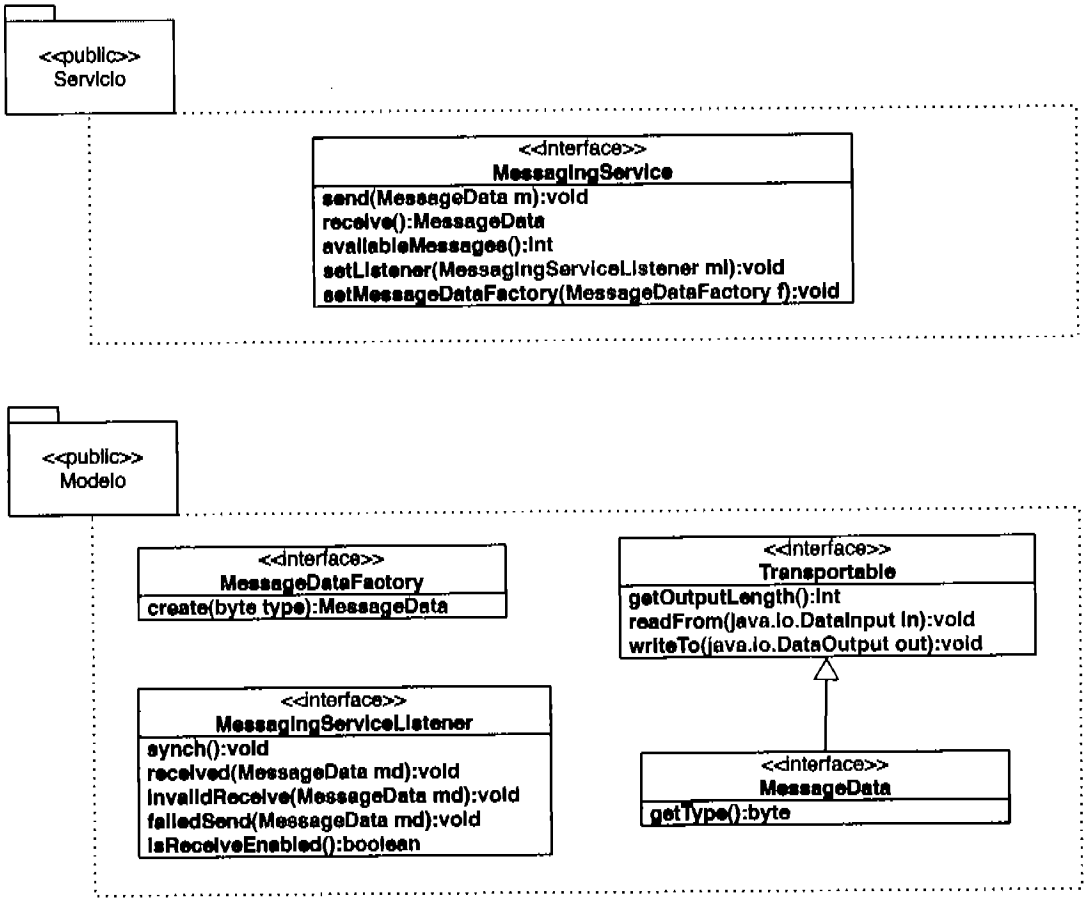


Figura 4.14: Diagrama de clases del modelo público del mensajero ROM, de acuerdo con la definición del UML; se indican adicionalmente los paquetes a los cuales corresponden las interfaces presentadas.

- *identificador proceso* (PID), un número entero de 4 bytes (int) que codifica el número IP (v4) del nodo, que es un número único en una red local de varios nodos;
- *número de secuencia*, un número entero de 8 bytes (long) el cual representa el contador de mensajes y, se puede utilizar para la detección de la omisión de un mensaje en un procesador (secuencia interrumpida).
- *tiempo*, un número entero de 8 bytes (long), codificando el tiempo UTC en milisegundos;
- *tipo de datos*, un indicador para el nivel de la aplicación, el cual permite distinguir diferentes tipos de datos que se transmiten adentro de un paquete ROM.

La composición de los mensajes previstos se muestra en la Figura (4.15).

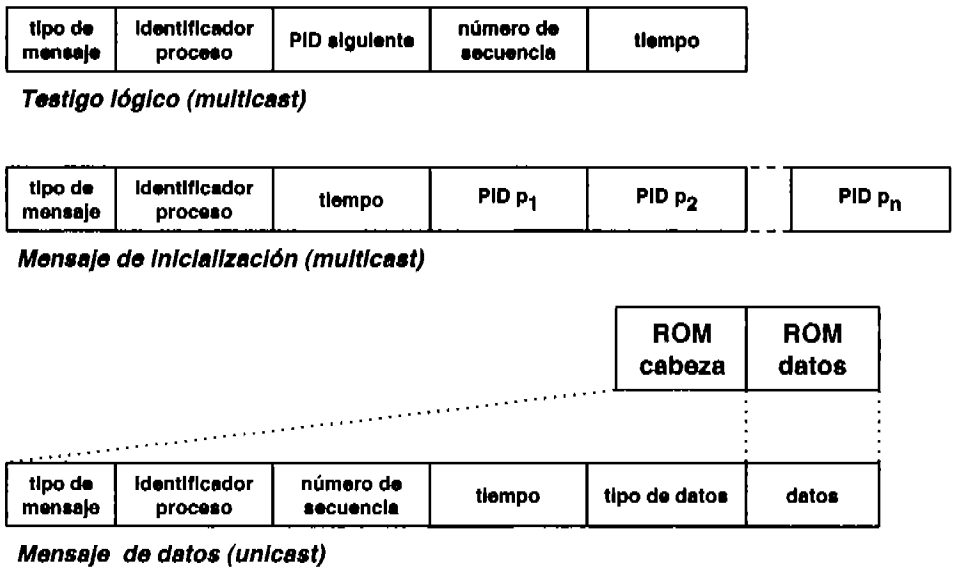


Figura 4.15: Paquetes del protocolo ROM

Todos los tipos primitivos se transmiten de la manera que especifica el `DataOutput` de la plataforma Java, con el orden de bytes para la transmisión en una red (típicamente big-endian).

4.3.3.6. Aspectos temporales

El tiempo máximo de una rotación del testigo lógico es

$$t_{rt} = n \cdot (t_{rm} + t_{mt}) \tag{4.1}$$

donde n, t_{rm} y t_{mt} son el número de procesadores en Π, el tiempo máximo para la rotación de un mensaje y el tiempo máximo para pasar el testigo lógico por medio de un multicast, respectivamente. Una rotación completa del testigo lógico significa que todos los procesadores tuvieron la posibilidad de enviar un mensaje. Sin embargo, es trivial extender este cálculo a un caso en el cual se permite la transmisión de más de un mensaje por procesador y turno. El valor de t_{rt} representa un límite superior en tiempo, ya que no necesariamente todos los nodos requieren enviar un mensaje en cada rotación del testigo lógico. Este límite se puede utilizar para la detección de la pérdida del testigo lógico cuando ocurre una falla total en el nodo, el cual posee el testigo lógico en ese momento.

El factor predominante en t_{rt} es t_{rm}, el tiempo para la rotación de un mensaje, el cual se puede expresar como

$$t_{rm} = n \cdot t_{um} \tag{4.2}$$

donde t_{um} es el tiempo para el unicast del mensaje de un nodo a otro. El valor de t_{rm} se puede utilizar por el nodo que espera la transmisión de un mensaje como límite superior en tiempo para la detección de la falla total de un nodo del anillo.

Para el sistema en tiempo real suave se propone utilizar un mecanismo para evaluar el tiempo t_{rm} máximo en el sistema al momento que se configura el anillo: el nodo que inicia el anillo circula k veces un mensaje de prueba con datos aleatorios, tomando el tiempo de cada rotación. El tiempo máximo de rotación de un mensaje se puede calcular de la siguiente manera:

$$t_{rm,max} = MAX(\hat{t}_{rm}) \cdot S \tag{4.3}$$

donde MAX,$\hat{t}_{rm}$ y S son una función que determina el valor máximo, el tiempo muestreado de la rotación y, un factor de seguridad que se debe elegir con base en los requisitos de la aplicación, respectivamente.

Bajo la suposición de que en el peor caso el multicast del testigo lógico tarda el tiempo máximo de la rotación de un mensaje por medio de unicasts, se puede calcular también un límite superior para t_{rt} utilizando la Ecuación (4.1).

4.3.4. Espacio de tuplas

4.3.4.1. Modelo básico

En el marco de este trabajo de investigación el TS se define como una memoria lógica, volátil y compartida, la cual puede almacenar un número arbitrario de tuplas sin conocer de antemano su estructura. Una tupla puede contener un número arbitrario de campos con un tipo primitivo. Por ejemplo $\{1; BP1; 1,94\}$ es una tupla con la estructura $\{int, string, float\}$.

Los tipos primitivos definidos para tuplas son:

1. *boolean*, *byte*, *short*, *int*, *long*, *float*, *double* y *String*, los cuales se definen de la misma manera que los tipos en el lenguaje Java por Gosling et al. (2005);

2. *mnemónica*, la cual representa una serie de tres bytes que codifican una mnemónica en ASCII para una salida o entrada, las cuales comúnmente se utilizan en SAOPs;

3. *binario*, el cual representa un conjunto de bytes y puede contener datos codificados por la aplicación.

Se permite la existencia de varios TS al mismo tiempo, lo cuál facilita al nivel de la aplicación el diseño de un SAOP con nodos que cuentan con recursos limitados de memoria física y/o únicamente requieren una parte de las variables del sistema.

4.3.4.2. Operación

El TS debe permitir 4 operaciones básicas síncronas:

1. *poner* una tupla nueva;
2. *actualizar* una tupla existente;
3. *tomar* una tupla existente de la memoria, removiéndola;
4. *leer* una tupla existente de la memoria, sin removerla.

Adicionalmente, para manejar eventos asíncronos, el TS debe permitir las siguientes dos operaciones:

1. *suscribir* a las actualizaciones de una tupla;
2. *cancelar* una suscripción.

Por diseño, existe la posibilidad de juntar una serie de operaciones básicas en una *transacción*, la cual tiene la siguiente forma algorítmica:

```
iniciar transacción
bloquear TS
ejecución de operaciones básicas
if SIN_FALLA
  then aplicar cambios
  else cancelar cambios
desbloquear TS
```

Se hace notar que existe un límite de operaciones básicas en una transacción si se requiere el transporte de la transacción en un solo paquete de datos. Este límite depende del MTU de la red física utilizada para el transporte de paquetes de datos entre nodos (Sección 4.3.2.1) y requiere de verificación en la implementación para la cantidad de datos esperados. Además, para asegurar la consistencia de los TS en todos los nodos, el acceso para manipulaciones se puede sincronizar con el testigo lógico del protocolo ROM.

4.3.4.3. Modelo público

El modelo público diseñado para el espacio de tuplas se muestra en el diagrama de paquetes en la Figura 4.16 y el diagrama de clases en la Figura 4.17.

4.3.4.4. Paquetes del protocolo

El protocolo del TS debe permitir la propagación de los cambios del contenido del TS a todos los nodos participantes. Las operaciones definidas anteriormente se traducen de la siguiente manera a acciones:

- *poner* una tupla a la acción *poner;*
- *actualizar* una tupla a la acción *actualizar;*

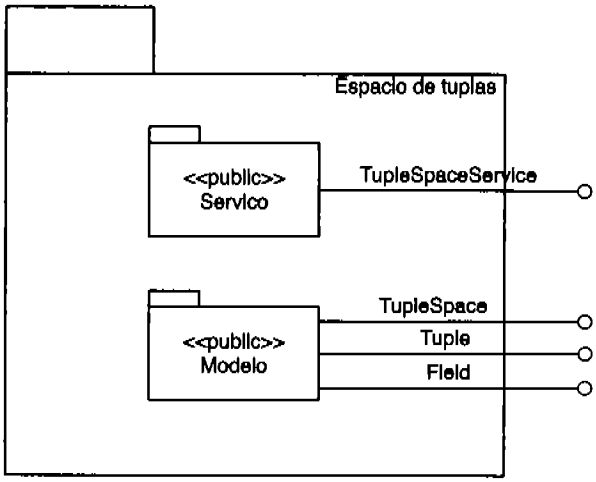


Figura 4.16: Diagrama de paquetes del modelo público del espacio de tuplas distribuido, de acuerdo con la definición del UML.

- *tomar* una tupla a la acción *remover;*
- *leer*, *suscribir* y *cancelar* son operaciones locales que no cambian el contenido del TS.

Los paquetes del protocolo siempre representan una transacción y están compuestos de los siguientes elementos:

- *identificador TS* , un byte que identifica el espacio de tuplas (256 posibles) al cual hay que aplicar la transacción;
- *número de acciones*, un número entero de 2 bytes (short), el cual representa el número de acciones en esta transacción;
- *tipo acción*, un byte que identifica el tipo de acción;
- *identificador tupla*, un número entero de 4 bytes (int) que identifica únicamente una tupla en un TS;
- *número de campos*, un número entero de 2 bytes (short), el cual representa el número de campos en una tupla;
- *tipo campo*, un byte que identifica el tipo de campo;
- *datos campo*, los cuales están definidos por el tipo del campo:
  - *boolean*, *byte*, *short*, *int*, *long*, *float*, *double* y *String*, los cuales se transmiten de manera definida por el *DataOutput* en el lenguaje Java por Gosling et al. (2005);
  - *mnemónica*, la cual se transmite como una serie de tres bytes que codifican letras en ASCII;
  - *binario*, el cual se transmite con el número de bytes codificado en un número entero de 2 bytes (short), seguido por los bytes que forman el contenido del campo.

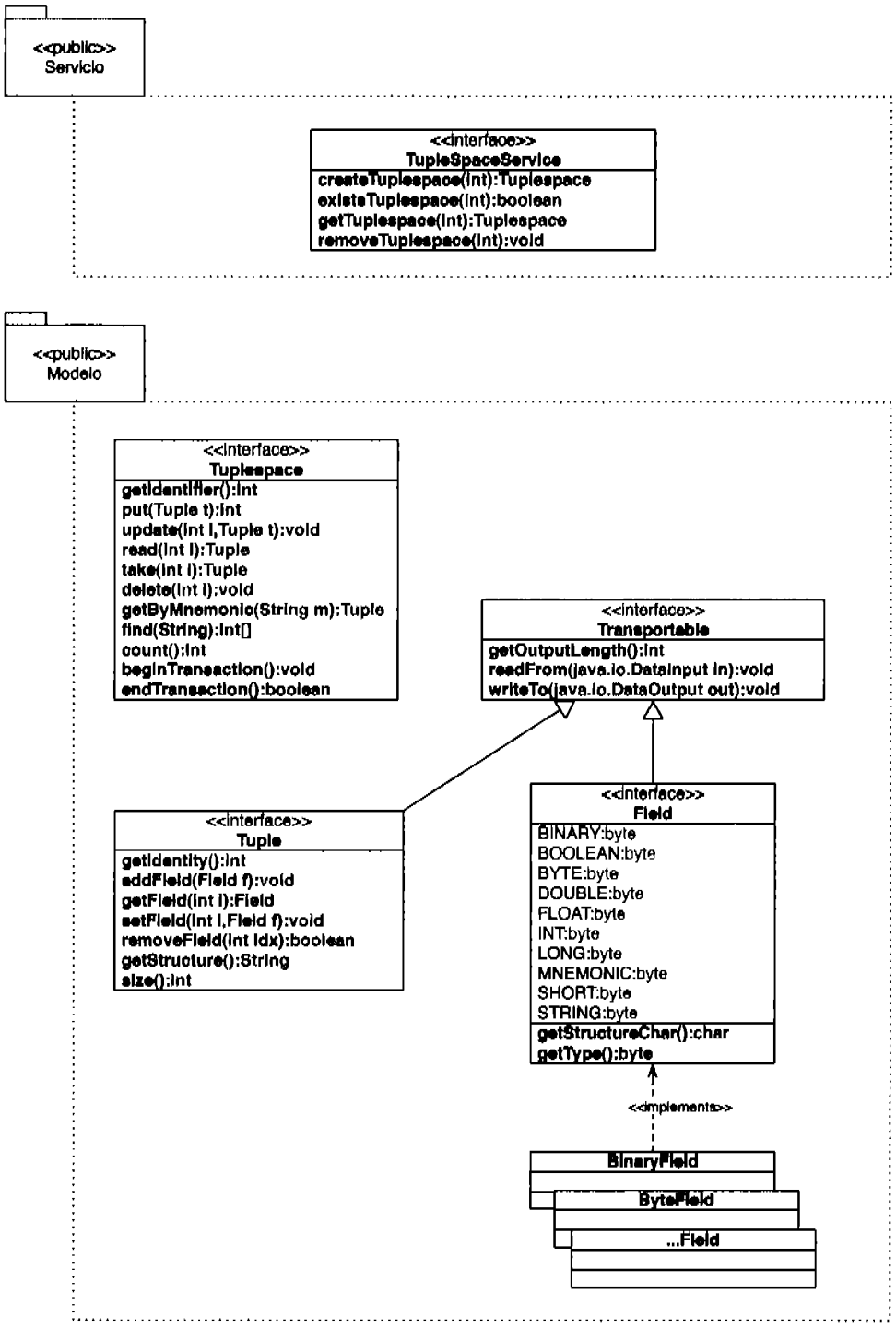


Figura 4.17: Diagrama de clases del módelo público del espacio de tuplas distribuido, de acuerdo con la definición del UML; se indican adicionalmente los paquetes a los cuales corresponden las interfaces presentadas.

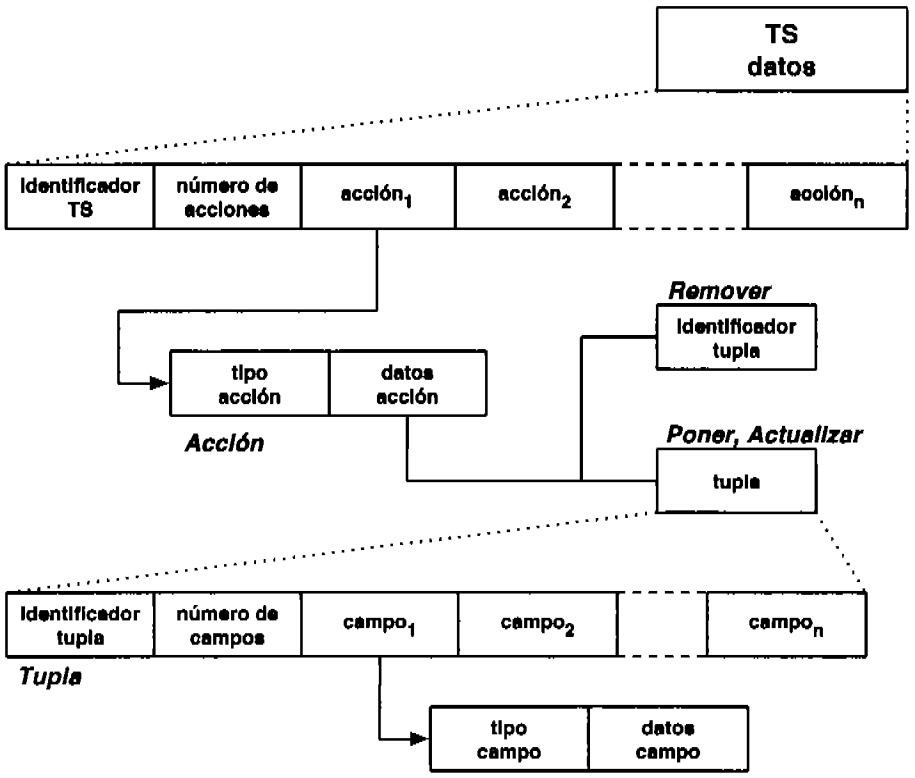


Figura 4.18: Paquete de datos del protocolo TS

La composición de los datos se muestra en la Figura (4.18).

Todos los tipos primitivos se transmiten de la manera que especifica el `DataOutput` de la plataforma Java, con el orden de bytes para la transmisión en una red (típicamente big-endian).

4.3.5. Protocolo NSD

4.3.5.1. Modelo básico y operación

El modelo básico del sistema consiste en un conjunto de nodos con componentes de software que deben de formar un sistema distribuido e interactuar por medio del intercambio de mensajes utilizando una red de área local. Los componentes que quieran interactuar pueden:

1. ofrecer un servicio de red (NS, por sus siglas en inglés Network Service);
2. usar un servicio ofrecido;
3. o las dos opciones anteriores al mismo tiempo.

Para establecer la comunicación, se requieren conocimientos explícitos de direcciones o números de puertas de los NSs y a veces de propiedades adicionales que describan el NS. El SD busca la autoconfiguración del sistema por medio de un NS que permita ofrecer y descubrir la información sobre otros servicios en la red de área local.

Para el descubrimiento de un servicio hay las siguientes operaciones:

1. *solicitud* y *respuesta*:

   a) un componente en un nodo busca un NS por medio del servicio NSD;

   b) el servicio NSD en el nodo solicita un NS con ciertas propiedades por medio de un multicast a todos los nodos en la red de área local;

   c) los servicios NSD en los nodos de la red responden por medio de un unicast, si tienen registrado un NS con propiedades que cumplan con la solicitud;

   d) el servicio NSD en el nodo solicitante informa al componente acerca de los resultados.

2. *anuncio:*

   a) un componente en un nodo quiere ofrecer un NS;

   b) utilizando el servicio NSD en el nodo, anuncia la disponibilidad del servico por medio de un multicast a todos los nodos en la red de área local.

4.3.5.2. Modelo público

El modelo público diseñado para el servicio NSD se muestra en el diagrama de paquetes en la Figura 4.19 y el diagrama de clases en la Figura 4.20.

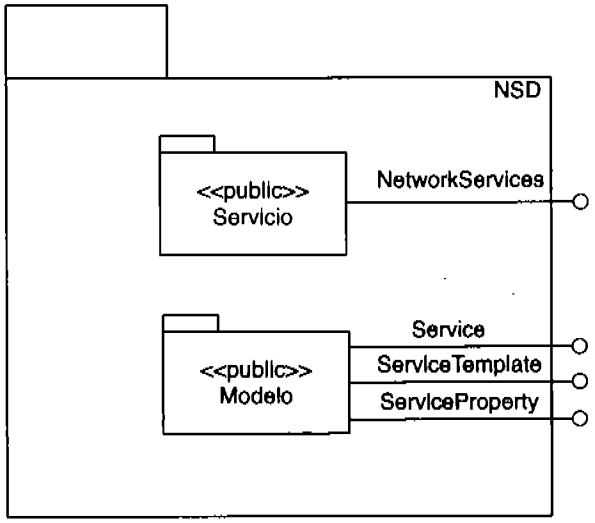


Figura 4.19: Diagrama de paquetes del modelo público del servicio NSD, de acuerdo con la definición del UML.

Este modelo público representa una abstracción que permite también implementaciones para los protocolos SLP2 (Guttman, 1999) y mDNS (Steinberg y Chesire, 2005).

4.3.5.3. Paquetes del protocolo

El protocolo del NSD debe permitir el intercambio de la información de los servicios de la red. Las operaciones definidas anteriormente, se traducen directamente a los siguientes tipos de mensajes:

1. *solicitud*

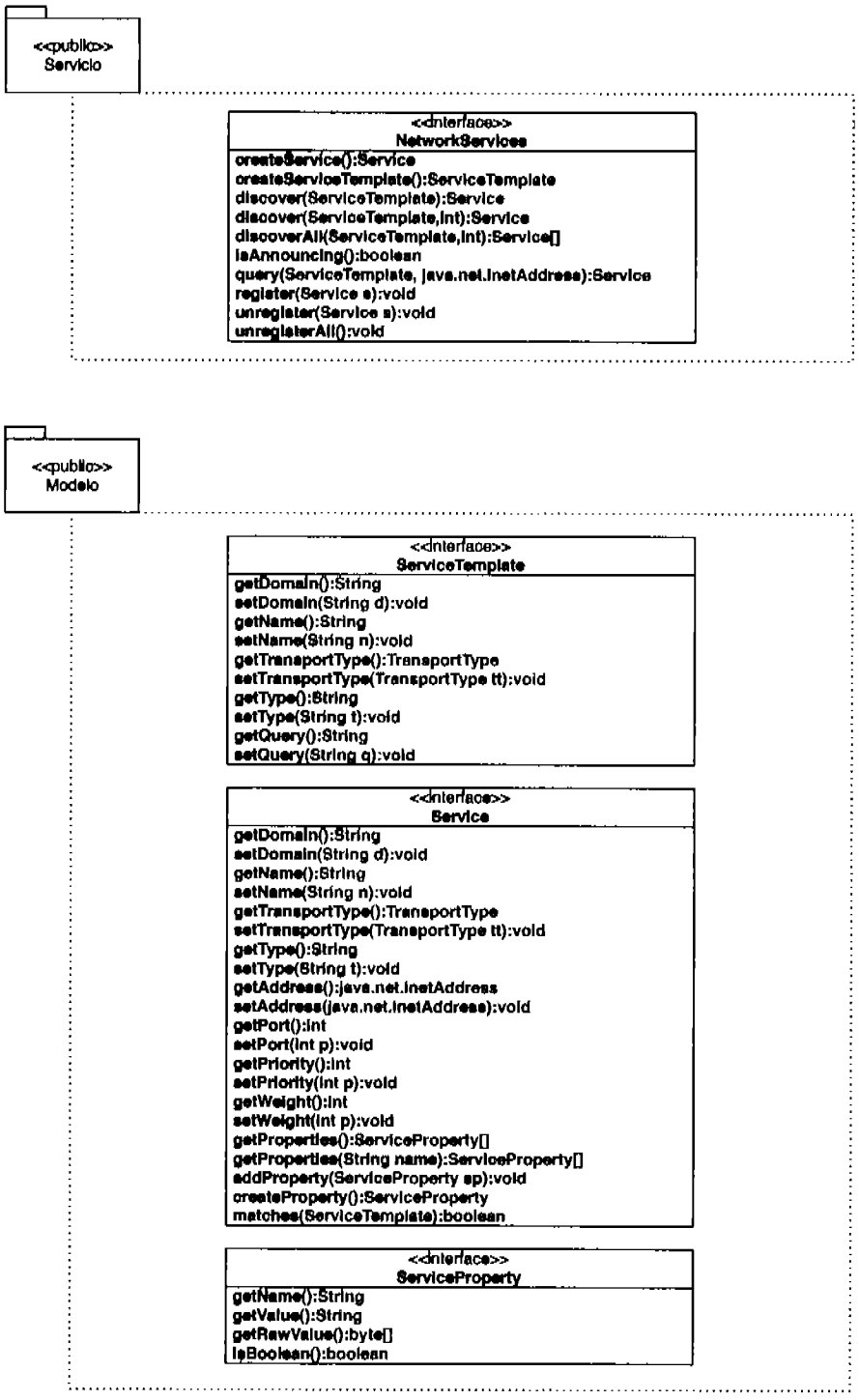


Figura 4.20: Diagrama de clases del módelo público del servicio NSD, de acuerdo con la definición del UML; se indican adicionalmente los paquetes a los cuales corresponden las interfaces presentadas.

2. *respuesta*

3. *anuncio de servicio disponible*

4. *anuncio de servicio no disponible*

Los paquetes del protocolo siempre representan uno de estos mensajes y están compuestos de los siguientes elementos:

- *tipo* mensaje, un byte que identifica los tipos de mensajes anteriormente descritos;
- *banderas*, dos bytes, los cuales se usan por el momento únicamente para indicar que hay mensajes adicionales siguientes (e.g. respuestas de múltiples paquetes por la cantidad de servicios encontrados);
- *identificador de mensaje*, un número entero de dos bytes, el cual identifica un mensaje y permite emparejar solicitudes con respuestas;
- *datos del mensaje*, los cuales pueden ser:
  - la plantilla de un servicio;
  - un servicio;
  - una lista de servicios.

La composición de los datos se muestra en la Figura (4.21).

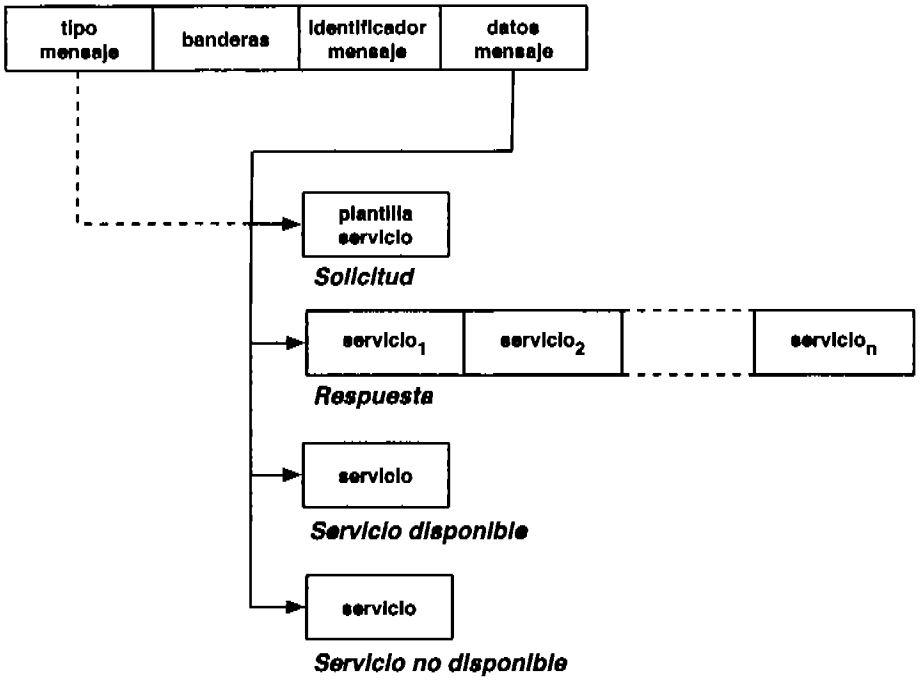


Figura 4.21: Paquetes del protocolo NSD

Los datos de los mensajes están compuestos de los siguientes elementos:

- *dominio*, un string;
- *tipo de transporte*, un byte que indica:
  - paquetes (e.g. UDP);
  - conexión (e.g. TCP);
- *tipo de servicio*, un string;
- *nombre de servicio*, un string;
- *solicitud de propiedades*, un string;
- *dirección IP*, un número entero de 4 bytes (int) que codifica el número IP (v4);
- *número de puerta*, un número entero de 2 bytes (short) que codifica el número de la puerta;
- *importancia*, un byte;
- *prioridad*, un byte;
- *número de propiedades*, un byte (e.g. máximo 255);
- *propiedad*, la cual consiste en:
  - *nombre*, un string;
  - *valor*, un string.

La composición de los datos se muestra en la Figura (4.22).

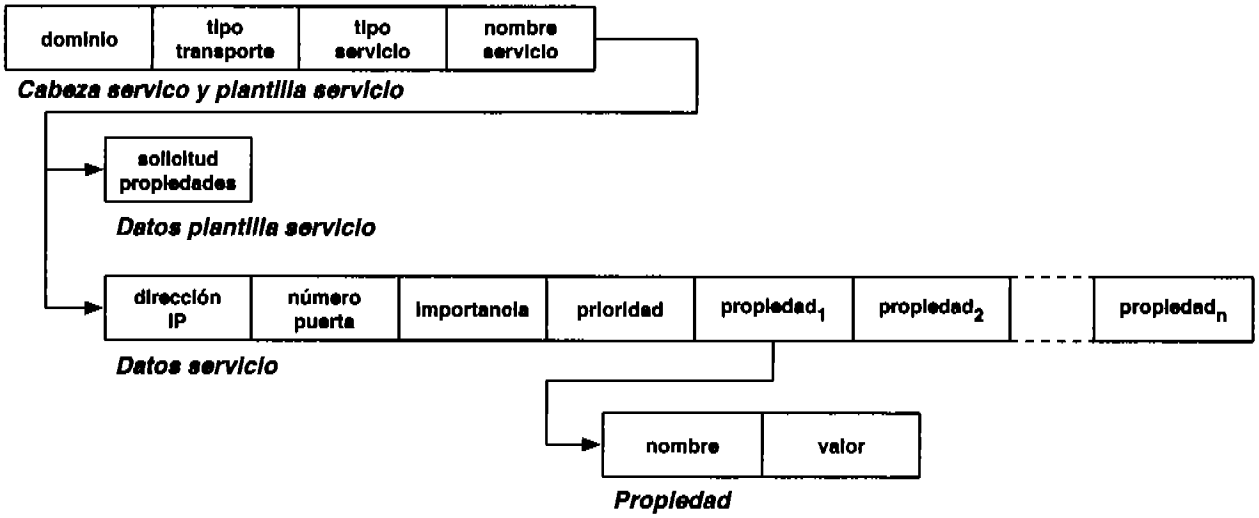


Figura 4.22: Datos de los paquetes del protocolo NSD

Todos los tipos primitivos se transmiten de la manera que especifica el *DataOutput* de la plataforma Java, con el orden de bytes para la transmisión en una red (típicamente big endian).

4.4. Implementación de la infraestructura y verificación

La implementación de los componentes de la infraestructura del SAOP se realizó en Java 2 y para la especificación de OSGi R4:

1. *Encuentro*, el componente informático que realiza el servico para descubrir servicios de la red de área local, de acuerdo con la solución propuesta en Sección 4.3.5.

2. *Tring*, el componente informático que realiza el mensajero ROM, de acuerdo con la solución propuesta en Sección 4.3.3.

3. *Retus*, el componente informático que realiza el espacio de tuplas distribuido, de acuerdo con la solución propuesta en Sección 4.3.4.

Para la verificación de la implementación se realizaron las siguientes pruebas:

1. Pruebas de entidades (veáse Apéndice D), para la funcionalidad básica de todos los elementos de las implementaciones;

2. Funcionalidad de los componentes informáticos de comunicación:

   a) pruebas del servicio de descubrimiento de servicios en la red de área local NSD;

   b) prueba del mensajero ROM que verifica el funcionamiento del mecanismo del protocolo y sus características temporales; y

   c) prueba del espacio de tuplas que verifica el funcionamiento del mecanismo y del protocolo para transmitir los cambios.

Para las pruebas en red de área local se utilizó el ambiente de pruebas descrito en el Apéndice E. Se hace notar que las implementaciones de *Tring* y *Retus* están basadas en una técnica de reciclaje de objetos para evitar la recolección de memoria durante el tiempo de corrida (Dautelle, 2004), lo cual permite su uso en ambientes de tiempo real suave.

El código fuente (incluyendo el código de las pruebas) y la documentación se pueden encontrar en el disco anexo a este trabajo. Para la validación de los componentes y el despliegue en un contenedor OSGi se utilizó la implementación de referencia de la especificación de OSGi R4 (Eclipse Foundation, 2007).

4.4.1. Encuentro (NSD)

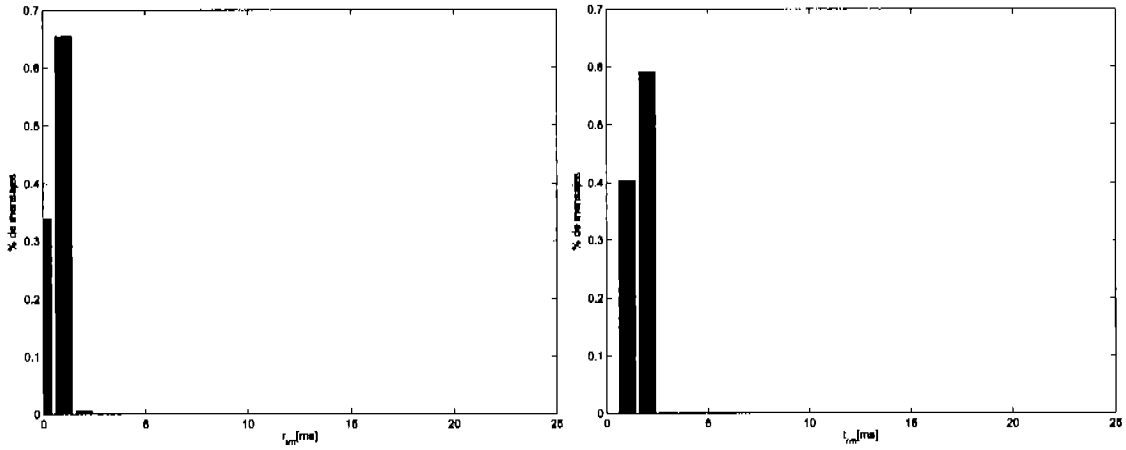
Para la verificación de la auto-configuración mediante el servicio NSD, se utilizó el ambiente de prueba (Apéndice E) y se comprobó la auto-configuración del anillo lógico para el mensajero ROM por medio del SD. Adicionalmente, se implementó una aplicación que puede monitorear todos los servicios ofrecidos en la red de área local por medio de *Encuentro*. Se hace notar que no se implementó un lenguaje para especificar propiedades de servicios cuando se solicitan a la red de área local. Aúnque está previsto en la especificación del protocolo para plantillas de servicios, este mecanismo no es realmente necesario para la implementación del prototipo, ya que no se requiere distinguir entre servicios equivalentes con propiedades distintas.

4.4.2. Tring (mensajero ROM)

Para la evaluación del comportamiento temporal del mensajero ROM en el ambiente de prueba (Apéndice E) se corrieron 10 repeticiones de un experimento con 30,000 circulaciones para un mensaje de:

1. 256 bytes de datos, los cuales son aproximadamente la cantidad máxima de datos en un SAOP de la escala del caso de estudio;
2. 1,280 bytes de datos, los cuales son aproximadamente la cantidad máxima de datos útiles que se pueden transmitir en un paquete.

La Figura 4.23 muestra un histograma de los tiempos de rotación del mensaje para cada tamaño. En los dos casos el tiempo máximo muestreado de rotación $\hat{t}_{rm}$ es de 20 ms.



(a) 256 bytes por mensaje

(b) 1280 bytes por mensaje

Figura 4.23: $t_{rm,i}$ para mensajes de 256 bytes (a) y 1,280 bytes (b) de datos, histograma de 10 repeticiones de un experimento con 30,000 rotaciones de mensajes.

A partir del máximo muestreado se puede calcular los límites de tiempo en el comportamiento temporal del anillo en el ambiente de prueba utilizando las ecuaciones presentadas en la Sección 4.3.3.6.

Los valores de $t_{rm,max}$ y de t_{rt} para varios factores de seguridad S se muestran en la Tabla 4.3. Con un factor de seguridad de $S = 5$, se puede suponer un tiempo máximo de rotación del testigo de 360 ms, lo permite cumplir con los requisitos para el proceso en el caso de estudio con un tiempo de muestreo alrededor de un $1s$, aún cuando se toma en cuenta la retransmisión de cada mensaje (i.e. límite presentado como $t_{rt,2}$, con $t_{rm} \cdot 2$ en Ec. 4.1).

Para Ethernet (100BASE-TX, 100 mbit) la tasa de transferencia teórica se puede calcular de la siguiente manera:

$$100 \cdot 1e6\ bits = 1e8\ bits \approx 1,220kb/s \tag{4.4}$$

La transferencia efectiva en términos de los datos que se puede calcular de los experimentos se muestra en Tabla 4.4. Se hace notar que la límitación no proviene de la cantidad de datos transmitidos siempre y cuando sean menor al MTU.

	$S=2$	$S=3$	$S=4$	$S=5$
$t_{rm,max}$	40	60	80	100
t_{rt} (n=3)	180	240	300	360
$t_{rt,2}$(n=3)	300	420	540	660

Tabla 4.3: Cálculo de los límites del comportamiento temporal de la implementación; los valores son tiempos en millisegundos.

Datos por mensaje	Tasa de transferencia
1,280 bytes	773.02 kb/s
256 bytes	364.37 kb/s

Tabla 4.4: Tasa de transferencia de datos

4.4.3. Retus (espacio de tuplas distribuido)

Para la verificación de la espacio de tuplas, se utilizó el ambiente de prueba (Apéndice E) y se comprobaron los mecanismos de:

1. *poner* una tupla nueva;
2. *actualizar* una tupla existente;
3. *tomar* una tupla existente de la memoria, removiéndola;
4. *leer* una tupla existente de la memoria, sin removerla;
5. *suscribir* a las actualizaciones de una tupla; y
6. *cancelar* una suscripción.

En particular, el mecanismo de suscripción se utilizó para verificar que se efectúan las operaciones en todos los nodos, y de ésta manera, comprobar la funcionalidad de la implementación del espacio de tuplas distribuido.

Capítulo 5

Prototipo de un SAOP para el caso de estudio

5.1. Introducción

El prototipo para el caso de estudio presenta la integración del diagnóstico de fallas y de la infraestructura para implementar un SAOP.

Normalmente el nivel de campo, mostrado en la Figura 4.2, conecta al SAOP con el proceso real, mediante:

1. varios sensores y actuadores;
2. un sistema de entradas y salidas eléctricas;
3. un componente del SAOP que forma la interfaz entre el sistema de entradas y salidas y el *middleware* del SAOP.

Sin embargo, en el caso de estudio debido al alto costo y a la inversión de tiempo que requieren los experimentos en el laboratorio, se realizó un componente que simula el nivel de campo (i.e. la infraestructura) y la dínamica del proceso.

El objetivo del prototipo es la verificación de la funcionalidad de un SAOP en el sistema distribuido descrito por el ambiente de pruebas (Apéndice E). La implementación debe permitir validar de manera integrada, tanto el diagnóstico, como la infraestructura, incluyendo:

1. el diseño y despliegue de componentes utilizando la arquitectura de software descrito en la Sección 4.3.1; y
2. la auto-configuración y el transporte de datos en tiempo real suave por medio del mecanismo de comunicación descrito en la Sección 4.3.2.

A continuación se presenta la arquitectura del prototipo, los detalles de la implementación de los componentes y el modelo y flujo de datos diseñado con base en el espacio de tuplas como *middleware*.

5.2. Arquitectura del prototipo

El prototipo consiste en los siguientes componentes:

1. la simulación del proceso;
2. el control secuencial para la operación del proceso;
3. el sensor de software para la estimación de la tasa de respiración; y
4. el diagnóstico de fallas con base en la clasificación de las características extraidas de la señal de respiración.

Desde la perspectiva lógica (funcional), la arquitectura del SAOP se basa en la infraestructura descrita en Capítulo 4 y se presenta en la Figura 5.1. El espacio de tuplas (Sección 4.3.4) actúa como *middleware* (Sección 4.3.2.2), cuya implementación maneja los detalles del intercambio de datos entre los componentes participantes.

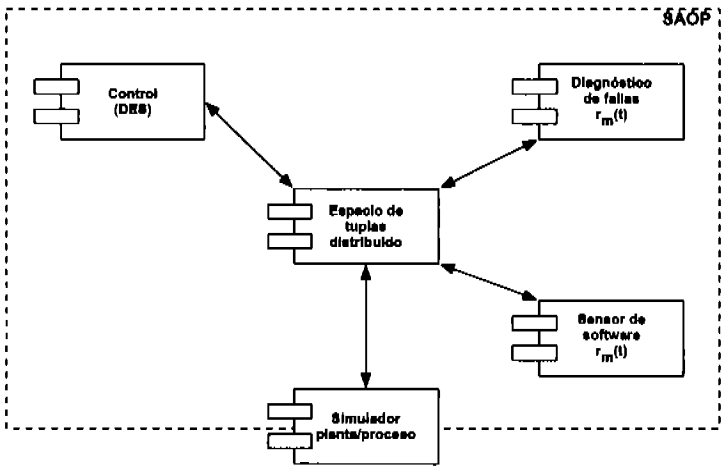


Figura 5.1: Perspectiva lógica (funcional) de la arquitectura del prototipo del SAOP

El diagrama de despliegue, mostrado en la Figura 5.2, describe la colocación de los componentes del prototipo en los nodos del ambiente de pruebas (Apéndice E).

5.3. Componentes del prototipo

5.3.1. Simulación del proceso (P)

La implementación del componente que simula el proceso realiza:

1. la simulación de los sensores y actuadores descritos en la Sección 2.2.1; y
2. la simulación del proceso dinámico dado por las Ec. (2.2) y (2.3).

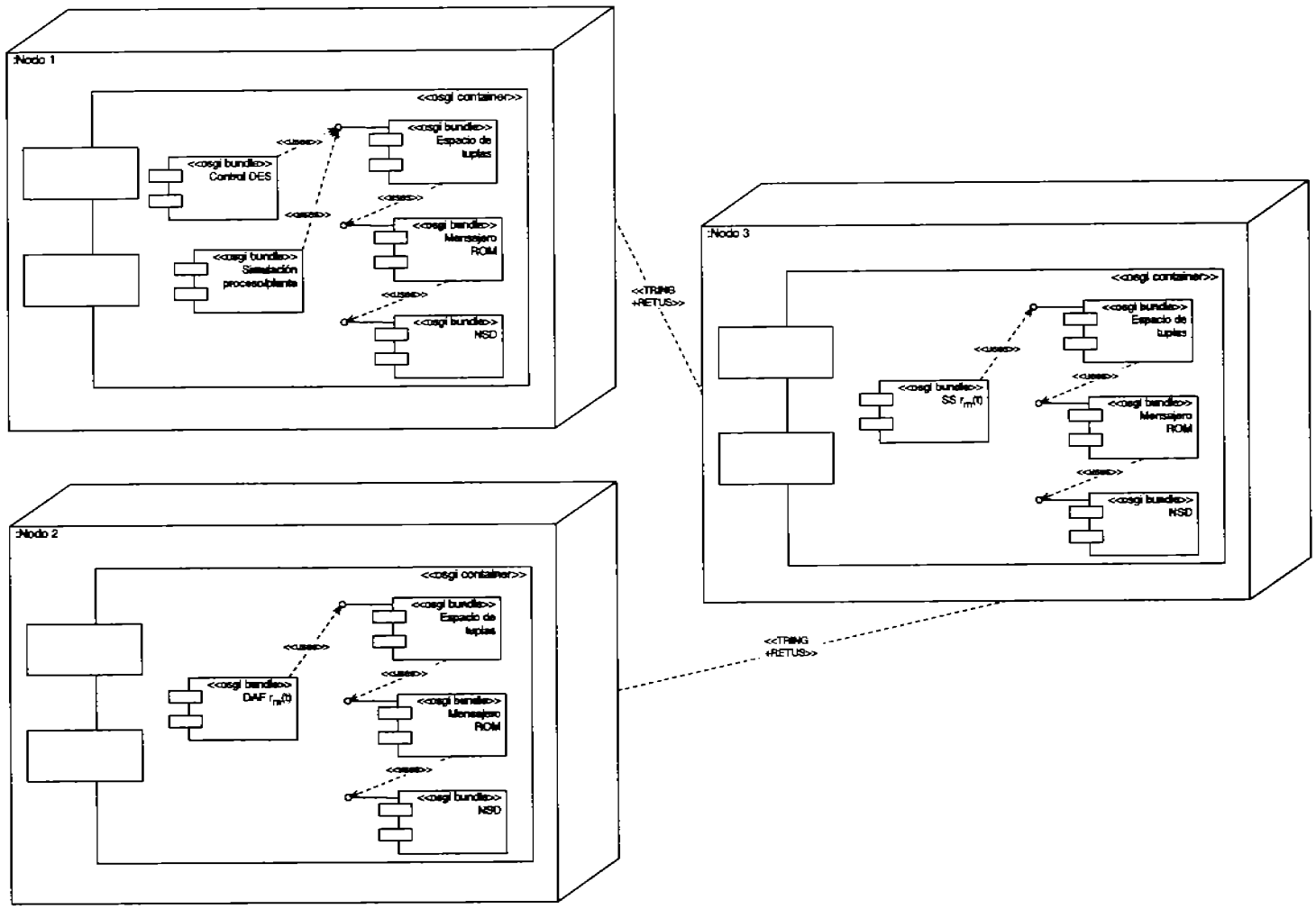


Figura 5.2: Diagrama de despliegue del prototipo del SAOP, de acuerdo con la definición de diagramas de despliegue del UML.

5.3.1.1. Sensores y actuadores

Los sensores y actuadores simulados por este componente se modelan con base en el modelo estructural del caso de estudio, descrito en la Sección 2.2.1. Para el sistema se pueden representar por medio de las entradas y salidas descritas en la Tabla 5.1.

Entradas	***Tipo***	***Mnemónica***
Interruptor bomba de entrada	*Interruptor*	SPI
Interruptor bomba de salida	*Interruptor*	SPO
Interruptor mezclador	*Interruptor*	SMI
Interruptor aireación	*Interruptor*	SAE

Salidas	***Tipo***	***Mnemónica***
Concentración oxígeno disuelto	*Medición*	OXY
Volumen	*Medición*	VOL
Flujo de entrada	*Medición*	QIN

Tabla 5.1: Tabla de entradas y salidas del componente que simula el proceso, desde la perspectiva del componente

5.3.1.2. Simulación del modelo dinámico

La simulación del modelo dinámico se realizó originalmente en MATLAB y, por lo tanto, para el prototipo se realizó una implementación en Java, con un algoritmo del método Runge Kutta de cuarto orden, lo cual presenta una solución númericamente robusta para resolver la integración de sistemas de ecuaciones diferenciales con un número de pasos fijos.

Para la verificación de la implementación en Java se utilizaron los resultados de la implementación en MATLAB y los de la implementación Java para los estados del sistema en el caso nominal, descritos por las condiciones iniciales de la Tabla 2.2 y, presentado en la Figura 2.5. La Tabla 5.2 presenta el promedio de los errores cuadrados entre las salidas de MATLAB y Java, los cuales indican que la implementación en Java se puede considerar básicamente equivalente a la de MATLAB.

Estado	m_x	m_s	m_o	V
MSE	0.001682	0.050934	0.000226	0.000003

Tabla 5.2: Promedios de los errores cuadrados entre los estados del sistema simulado en MATLAB y la implementación en Java

5.3.2. Control secuencial de la operación del proceso (C)

La implementación del componente de control secuencial realiza la operación del proceso, descrita en la Sección 2.2.4. Formalmente, la operación del proceso se puede modelar como un sistema de eventos discretos (DES), con un autómata con estados finitos (Cassandras, 1993). Este autómata, mostrado en la Figura 5.3, se puede definir con el conjunto (E, X, Γ, f, x_0):

$$
\begin{aligned}
E &= \{e_1, e_2, e_3, e_4, e_5\} \\
X &= \{q_0, q_1, q_2, q_3, q_4\} \\
\Gamma &= \Gamma(q_0) = \{e_1\}, \Gamma(q_1) = \{e_2\}, \Gamma(q_2) = \{e_3\}, \\
&\quad \Gamma(q_3) = \{e_4\}, \Gamma(q_4) = \{e_5\} \\
f &= f(q_0, e_1) = q_1, f(q_1, e_2) = q_2, f(q_2, e_3) = q_3, \\
&\quad f(q_3, e_4) = q_4, f(q_4, e_5) = q_0, \\
x_0 &= q_0
\end{aligned} \tag{5.1}
$$

donde E, X, Γ, f y x_0 son el conjunto de eventos, el conjunto de estados posibles, los eventos posibles en cada estado, el conjunto de funciones de transición y el estado inicial, respectivamente.

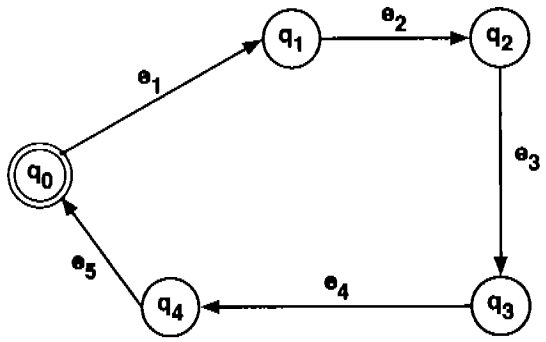


Figura 5.3: DES de la operación básica del proceso

En el SAOP el componente de control secuencial tiene la función de realizar la operación por medio de:

1. la generación de los eventos del conjunto E; y

2. la manipulación de los actuadores, de acuerdo con las funciones de transición.

La implementación utiliza un esquema simple con tiempos fijos (Betancur et al., 2005), y actúa sobre las salidas descritas en la Tabla 5.3, las cuales coinciden con las entradas del componente de la simulación del proceso descritas en la Tabla 5.1.

Salidas	***Tipo***	***Mnemónica***
Interruptor bomba de entrada	*Interruptor*	SPI
Interruptor bomba de salida	*Interruptor*	SPO
Interruptor mezclador	*Interruptor*	SMI
Interruptor aireación	*Interruptor*	SAE
Estado del proceso	*Estado*	PST

Tabla 5.3: Tabla de salidas del componente de control, desde la perspectiva del componente

5.3.3. Sensor de software de la tasa de respiración (SS)

La implementación del componente del sensor de software realiza la estimación de la tasa de respiración basada en el observador descrito en la Sección 3.2.2.2.

Este observador se realizó originalmente en MATLAB y, por lo tanto, para el prototipo se realizó una implementación en Java. Para su validación se utilizaron los resultados de la implementación en MATLAB, y los de la implementación Java, del sistema en el caso nominal, descrito por las condiciones iniciales de la Tabla 2.2 y, presentado en la Figura 2.5.

El promedio de los errores cuadrados entre las salidas de MATLAB y Java para el caso nominal, descrito por las condiciones iniciales de la Tabla 2.2 y presentado en la Figura 2.5, es de 0.144344. Es un poco más alto que en el caso de la simulación del proceso dinámico, sin embargo, para la verificación es un error tolerable, lo cuál se comprobó por medio de la clasificación de la señal utilizando los clasificadores entrenados en MATLAB para la validación de la solución del diagnóstico de fallas presentada en la Sección 3.5.2.

La implementación utiliza las entradas descritas en la Tabla 5.4, las cuales coinciden en su mayoría con las salidas del componente de la simulación del proceso descritas en la Tabla 5.1, y produce como salida el valor de la tasa de respiración.

Entradas	***Tipo***	***Mnemónica***
Estado del proceso	Estado	PST
Concentración oxígeno disuelto	*Medición*	OXY
Volumen	*Medición*	VOL
Flujo de entrada	*Medición*	QIN

Salidas	***Tipo***	***Mnemónica***
Tasa de respiración	*Medición*	RES

Tabla 5.4: Tabla de salidas del componente del sensor de software, desde la perspectiva del componente

5.3.4. Diagnóstico de fallas (D)

La implementación del componente del diagnóstico de fallas realiza el diagnóstico basado en la clasificación de las características de la señal de la tasa de respiración descrita y formalizado en la Sección 3.4.2.

La extracción de las características de la señal de respiración se realizó originalmente en MATLAB y, por lo tanto, para el prototipo se realizó una implementación en Java. Para su validación se utilizaron resultados de la implementación del simulador en MATLAB, y se verificó que los valores de las características extraídas por el algoritmo realizado en MATLAB y en Java sean equivalentes para la señal de respiración en el caso nominal, descrita por las condiciones iniciales en la Tabla 2.2 y, presentada en la Figura 2.5. Los clasificadores de WEKA se pueden utilizar sin cambios, dado que WEKA está realizado en Java.

La implementación utiliza como entrada la tasa de respiración, y produce como salida el diagnóstico, mediante la aplicación de los mismos clasificadores que se utilizaron para la validación en el Capítulo 3. El resumen de entradas y salidas de este componente se presenta en la Tabla 5.5.

Entradas	*Tipo*	*Mnemónica*
Estado del proceso	Estado	PST
Tasa de respiración	*Medición*	RES

Salidas	*Tipo*	*Mnemónica*
Diagnóstico	*Estado*	DIA

Tabla 5.5: Tabla de entradas y salidas del componente del diagnóstico de fallas, desde la perspectiva del componente

5.4. Modelo y flujo de datos

De las Tablas 5.1, 5.3, 5.4 y 5.5 se pueden extraer todas las variables que requieren de una especificación para el intercambio entre los diferentes componentes del prototipo. Estos se presentan en la Tabla 5.6.

Variable	*Tipo*	*Mnemónica*
Interruptor bomba de entrada	*Interruptor*	SPI
Interruptor bomba de salida	*Interruptor*	SPO
Interruptor mezclador	*Interruptor*	SMI
Interruptor aireación	*Interruptor*	SAE
Estado del proceso	*Estado*	PST
Diagnóstico	*Estado*	DIA
Concentración del oxígeno disuelto	*Medición*	OXY
Volumen	*Medición*	VOL
Flujo de entrada	*Medición*	QIN
Tasa de respiración	*Medición*	RES

Tabla 5.6: Tabla de las variables intercambiadas en el prototipo

El flujo de variables entre los componentes del prototipo, sin tomar en cuenta el espacio de tuplas o la distribución de los componentes, se muestran en la Figura 5.4.

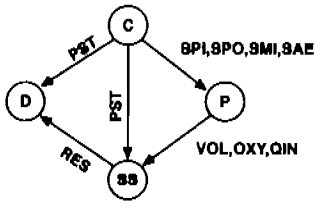


Figura 5.4: Flujo de variables entre componentes del prototipo, sin tomar en cuenta el espacio de tuplas o la distribución de los componentes en diferentes nodos (C representa el control, D el diagnóstico, P la simulación del proceso, y SS el sensor de software).

Todo el intercambio de datos entre componentes está realizado con base en el modelo del espacio de tuplas (Sección 4.3.4). La Tabla 5.7 presenta la definición de las tuplas utilizadas para las entradas y salidas de los componentes presentados anteriormente.

Los resultados de los tiempos presentados para Tring en la Sección 4.4.2 son válidos para el prototipo, considerando que:

Tupla	Campo 1	Campo 2
Interruptor	*mnemónica*	*boolean*
Estado	*mnemónica*	*int*
Medición	*mnemónica*	*double*

Tabla 5.7: Definiciones de las tuplas para modelar las entradas y salidas de los componentes

1. el sistema de prueba es equivalente;
2. la secuencia en la rotación del testigo lógico es la que se presenta en la Figura 5.5; y
3. el factor de seguridad $S = 5$ es suficiente para tomar en cuenta el procesamiento en los componentes participantes.

El orden particular de los nodos se configuró por medio de la propiedad de prioridad definida en el protocolo NSD (Sección 4.3.5).

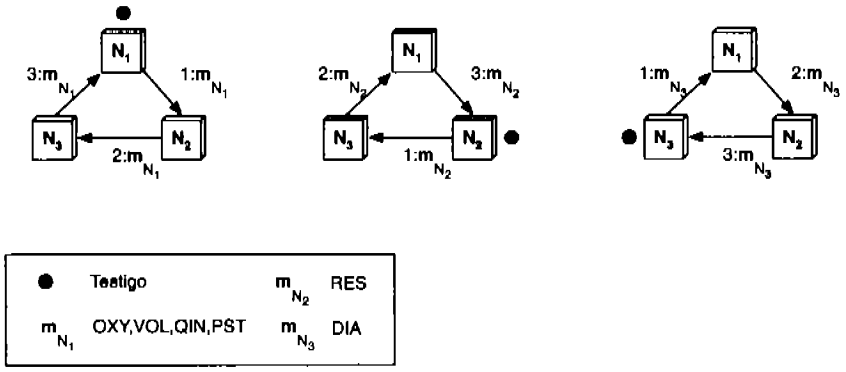


Figura 5.5: Secuencia de una circulación del testigo lógico en el anillo del prototipo

5.4.1. Verificación de la operación del prototipo

Se comprobó con 10 experimentos la funcionalidad de la implementación del prototipo, incluyendo la operación (i.e. control de la simulación) y el diagnóstico correspondiente a los parámetros utilizados para la corrida. Estos experimentos comprobaron el diagnóstico del proceso, con el clasificador de 5 clases para:

1. desviaciones del parámetro K_i, aumentando y disminuyendo el valor en 20 %;
2. desviaciones del parámetro K_s, aumentando y disminuyendo el valor en 20 %; y
3. la condición normal, con valores de K_i y K_s nominales.

Para cada experimento se realizó una repetición, lo cual da un total de 18h de corrida. El resumen de los experimentos y el resultado del diagnóstico se presenta en la Tabla 5.8.

#	Parámetro	Diagnóstico	t
1	$K_i \cdot 0.8$	$K_i \downarrow$	1.8
2	$K_i \cdot 0.8$	$K_i \downarrow$	1.8
3	$K_i \cdot 1.2$	$K_i \uparrow$	1.8
4	$K_i \cdot 1.2$	$K_i \uparrow$	1.8
5	$K_s \cdot 0.8$	$K_s \downarrow$	1.8
6	$K_s \cdot 0.8$	$K_s \downarrow$	1.8
7	$K_s \cdot 1.2$	$K_s \uparrow$	1.8
8	$K_s \cdot 1.2$	$K_s \uparrow$	1.8
9	*normal*	*normal*	1.8
10	*normal*	*normal*	1.8

Tabla 5.8: Experimentos para verificar la operación del prototipo

Capítulo 6

Conclusiones

6.1. Conclusión

El presente trabajo, engloba e integra una investigación interdisciplinaria en las áreas de:

- diagnóstico de fallas;
- análisis y diseño orientado a objetos;
- arquitectura de sistemas de software;
- redes y sistemas distribuidos; y
- procesos biológicos discontinuos para el tratamiento de aguas residuales.

Resuelve de manera integral el problema que significa crear un sistema automático de operación con un nivel de supervisión con diagnóstico de fallas, el cual:

1. asegura la eficiencia y confiabilidad en el funcionamiento de un proceso; y
2. puede aplicarse a sistemas de pequeña escala, donde el costo de inversión es un factor importante para el diseño.

Se proponen soluciones nuevas y genéricas, las cuales pueden aplicarse a una familia de problemas con características similares a las resumidas en la Sección 2.3. Se verificó la potencialidad de las propuestas, mediante la aplicación a un caso de estudio: un proceso biológico para el tratamiento de aguas residuales en ambientes urbanos e industriales pequeños.
En la parte del diagnóstico de fallas, presentado en el Capítulo 3, la principal contribución de este trabajo fue resolver de manera genérica la problemática de la evaluación de la viabilidad para resolver un problema de detección y aislamiento de fallas (DAF) con un enfoque y método:

1. de modo *a priori*, sin requerir la aplicación del método de detección y aislamiento de fallas; y
2. con un número restringido de sensores.

Se presentan procedimientos adecuados para la evaluación de la viabilidad de un diagnóstico en dos enfoques principales:

1. el enfoque basado en modelos analíticos; y
2. el enfoque basado en modelos con base en datos o señales.

Estos procedimientos se aplicaron para resolver el problema del diagnóstico en el caso de estudio. Para el enfoque con base en señales en particular, se encontró una solución novedosa, que consiste en la integración de un análisis de sensibilidad, un método de extracción de características y un método de clasificación para la obtención de la detección y el aislamiento de fallas definidas como desviaciones en parámetros del modelo del proceso.

En la parte de la realización de un sistema automático de operación, la contribución en este trabajo de investigación fue resolver de manera genérica la problemática de una arquitectura modular, integrada y abierta, la cual:

1. se realiza, de acuerdo con el desarrollo actual del área, en su mayor parte con software;
2. permite el manejo de la complejidad en el desarrollo y despliegue, como sistema distribuido;
3. toma en cuenta la necesidad de estos sistemas de funcionar en tiempo real, de acuerdo con las propiedades temporales del proceso; y
4. utiliza una infraestructura que no requiere de una inversión inicial muy grande.

Por un lado se presenta una arquitectura básica de software, basado en OSGi, la cual principalmente aporta la modularidad para el desarrollo y el despliegue en nodos distribuidos. Esta arquitectura es consecuente con las perspectivas importantes para el desarrollo de un SAOP, que parte de un problema teórico, el cual proporciona una cierta funcionalidad al sistema y se debe realizar como un componente informático.

Por otro lado se atacó el problema de la comunicación entre componentes, como parte de un sistema distribuido, en el cual no importa la ubicación física de los componentes. En el Capítulo 4, se presentan mecanismos de comunicación especificados como protocolos y modelos públicos de servicios, que operan sobre una red de área local Ethernet, que:

1. permiten la integración de componentes realizados para otras plataformas y arquitecturas;
2. permiten el transporte de datos en tiempo real, por lo menos de acuerdo con las necesidades temporales del proceso del caso de estudio;
3. consideran la problemática de la configuración de un sistema distribuido en la práctica; y
4. por el uso de Ethernet al nivel físico, presentan una solución de bajo costo y de alta disponibilidad.

Estos mecanismos de comunicación y sus atributos se comprueban por medio de una implementación particular, llevando a cabo pruebas de entidad y experimentos de comunicación.

De estas dos partes, la contribución es una infraestructura para el desarrollo y la realización de un SAOP, que permite una operación autónoma, eficiente y de bajo costo, en tiempo real adecuado para el proceso, y con una integración que permite un nivel de supervisión con diagnóstico de fallas, el cuál pueda prevenir un impacto económico y ambiental en caso de un comportamiento anormal.

Finalmente, se presenta también el prototipo de un SAOP para el caso de estudio, el cuál se realizó con la infraestructura e incluye la realización de un nivel de diagnóstico de fallas basado en la

solución para la parte del DAF. Este prototipo se utilizó exitosamente para verificar la funcionalidad de la infraestructura, sin embargo, debido al alto costo y a la inversión de tiempo que requieren los experimentos en el laboratorio, se reemplazó todo el nivel de campo y el proceso, con un componente que realiza una simulación de los sensores y actuadores, y la dinámica del proceso.

6.2. Trabajo futuro

En la parte del diagnóstico de fallas existe la posibilidad de la aplicación del procedimiento de la evaluación de la viabilidad, y de la metodología para el enfoque basado en señales, a otros casos con características similares a las que se encuentran resumidas en la Sección 2.3. Estos casos podrían contar con la disponibilidad de más señales, y por lo tanto, más características correspondientes, lo cual abre la posibilidad de investigar métodos para la reducción del conjunto de características por medio de algoritmos para la selección de las características más relevantes, sin afectar el desempeño del diagnóstico.

En la parte de la infraestructura, se podría estudiar el comportamiento temporal para otros casos que requieren una comunicación:

1. en tiempo real suave con tiempos de ciclo menor a un segundo; y
2. en tiempo real duro.

En estos casos se va a requerir un ambiente de pruebas mas homogéneo en cuanto a los nodos participantes, y en caso de tiempo real duro, hardware y software especial. El hardware se refiere a la parte física de la red y los nodos; y software a sistemas operativos en tiempo real, RTOS (por sus siglas en ingles real time operating system), y máquinas virtuales de Java que implementan la especificación de Java en tiempo real. Como alternativa también existe la posibilidad de un desarrollo de los mismos servicios y protocolos en otras plataformas (i.e. no para una máquina virtual), lo cuál podría permitir optimizaciones específicas para mejorar el desempeño temporal del mecanismo de comunicación.

Finalmente, existe la posibilidad de utilizar el protocolo para el descubrimiento de servicios (NSD) de manera independiente del mecanismo de comunicación y en contextos ajenos al caso de un SAOP. El uso en otros contextos, especialmente cuando son menos restrictivos, podrían requerir de extensiones del protocolo o de la implementación del servicio NSD. Por ejemplo, con un lenguaje que especifique de manera genérica filtros para propiedades de servicios solicitados.

Bibliografía

Andrews, J., 1968. A mathematical model for the continous culture of microorganisms utilizing inhibiting substrates. Biotechnol. Bioeng. 10, 707–723.

Åström, K., Wittenmark, B., 1984. Computer Controlled Systems. Prentice Hall, Englewood Cliffs, UK.

Bastin, G., Dochain, D., 1990. On-line Estimation and Adaptive Control of Bioreactors. Process measurement and control. Elsevier Science Publishers, Amsterdam, Netherlands, ISBN: 0444884300 (U.S.).

Betancur, M., Dochain, D., Fibrianto, H., 2004. Validated model. EOLI Deliverable 2.3, European Community Research (ICA4-CT- 2002-10012).

Betancur, M., Moreno-Andrade, I., Moreno, J., Dochain, D., Buitrón, G., July 2005. Modeling for the optimal biodegradation of toxic wastewater in a discontinuous reactor. En: Microbial Population Dynamics in Biological Wastewater Treatment (ASPD4). Queensland, Australia.

Blanke, M., Kinnaert, M., Lunze, J., Staroswiecki, M., 2003. Diagnosis and Fault-Tolerant Control, 1o. Edición. Springer Verlag, Berlin, Germany, ISBN: 3-540-01056-4.

Bollela, G., Graham, S., Lehman, T. J., 1999. Real-time tspaces. En: IECON '99 Proceedings. Vol. 2. Industrial Electronics Society, pp. 837–842.

Cassandras, C. G., 1993. Discrete Event Systems - Modelling and Performance Analysis. Aksen Associates.

Charbonnier, S., Garcia-Beltran, C., Cadet, C., Gentil, S., February 2005. Trends extraction and analysis for complex system monitoring and decision support. Engineering Applications of Artificial Intelligence 18 (1), 21–36.

Cheshire, S., Aboba, B., Guttman, E., May 2005. Dynamic configuration of ipv4 link-local addresses. URL `http://files.zeroconf.org/rfc3927.txt`

Daims, H., Nielsen, J. L., Nielsen, P., Schleifer, K.-H., Wagner, M., 2001. In situ characterization of nitrospira-like nitrite oxidizing bacteria active in wastewater treatment plants. App. Env. Microbiol. 67 (11), 5273–5284.

Dash, S., Venkatasubramanian, V., 2000. Challenges in the industrial applications of fault diagnostic systems. Computers and Chemical Engineering 24, 785–791.

Dautelle, J.-M., September 2004. Javolution.
URL `http://javolution.org`

Défago, X., Schiper, A., Urbán, P., 2004. Total order broadcast and multicast algorithms: Taxonomy and survey. ACM Comput. Surv. 36 (4), 372–421.

Eaton, J. W., 2002. GNU Octave Manual. Network Theory Limited, Bristol, UK, ISBN: 0-9541617-2-6.
URL http://www.octave.org

Eclipse Foundation, 2007. Equinox osgi r4.
URL http://www.eclipse.org/equinox/

EPA, U., 1999. Wastewater Technology Fact Sheet - Sequencing Batch Reactors. EPA 832-F-99-073.

Frank, P. M., 1978. Introduction to System Sensitivity Theory. Academic Press, New York, NY, USA, ISBN: 0-12-265650-4.

Gamma, E., Beck, K., 2000. junit regression testing framework.
URL http://www.junit.org

Gamma, E., Helm, R., Johnson, R., Vlissides, J., January 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Professional Computing Series. Addison-Wesley.

Gelernter, D., December 1992. Mirror Worlds, reprint Edición. Oxford University Press.

Gogate, P. R., Pandit, A. B., 1999. Survey of measurement techniques for gas-liquid mass transfer coefficient in bioreactors. Biochemical Engineering Journal 4, 7–15.

Gosling, J., Joy, B., Steele, G., Bracha, G., June 2005. The Java Language Specification, 3o. Edición. Java Series. Addison-Wesley Prodessional.

Guttman, E., 1999. Service location protocol: Automatic discovery of ip network services. IEEE Internet Computing 3 (4), 71–80.

Halang, W. A., Sanz, R., Babuska, R., Roth, H., 2006. Information and communication technology embraces control: Status report prepared by the ifac coordinating committee on computers, cognition and communication. Annual Reviews in Control In Press.

Hangos, K., Cameron, I., 2001. Process Modelling and Model Analysis. Academic Press.

Heck, B. S., Wills, L. M., Vachtsevanos, G. J., February 2003. Software technology for implementing reusable, distributed control systems. IEEE Control Systems Magazine, 21–35.

Heineman, G., Councill, W., 2001. Component-Based Software Engineering: Putting the Pieces Together. Addison Wesley, Reading, MA.

Helal, S., July-Sept. 2002. Standards for service discovery and delivery. IEEE Pervasive Computing 1 (3), 95–100.

Henze, M., Gujer, W., Mino, T., van Loosdrecht, M., 2000. Activated sludge models ASM1, ASM2, ASM2d and ASM3. Scientific and Technical Report 9, International Water Association, ISBN:1900222248.
URL http://www.iwapublishing.com/template.cfm?name=isbn1900222248

Irvine, R. L., Ketchum, L. H. J., 1988. Sequencing batch reactor for biological wastewater treatment. Critical Reviews in Environmental Control 18, 225–294.

Isermann, R., 1997. Supervision, fault-detection and fault-diagnosis methods - an introduction. Control Eng. Practice 5 (5), 639–652.

Isermann, R., 2000. Integration of fault detection and diagnosis methods. En: Patton, R. J., Frank, P. M., Clark, R. N. (Eds.), Issues of Fault Diagnosis for Dynamic Systems. Springer Verlag, London, pp. 15–45.

Isermann, R., 2005. Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance. Springer Verlag.

Isermann, R., Ballé, P., 1997. Trends in the application of model-based fault detection and diagnosis of technical processes. Control Eng. Practice 5 (5), 709–719.

Jasperneite, J., Watson, K., 2003. Bestimmung von oberen zeitschranken in ethernet-netzwerken.

Ketchum, L. H. J., 1997. Design and physical features of sequencing batch reactors. Water Science and Technology 35, 11–18.

Kourti, T., 2003. Multivariate dynamic data modeling for analysis and statistical process control of batch processes, start-ups and grade transitions. Journal of Chemometrics 17 (1), 93–109.

Larman, C., 2003. UML y Patrones, segunda Edición. Pearson Educación, Madrid, España.

Leis, J. R., Kramer, M. A., 1988. The simultaneous solution and sensitivity analysis of systems described by ordinary differential equations. ACM Trans. Math. Softw. 14 (1), 45–60.

Lindberg, C.-F., Carlson, B., 1996. Estimation of the respiration rate and oxygen transfer function utilizing a slow do sensor. Wat. Sci. Tech. 33 (1), 325–333.

Lindholm, T., Yellin, F., April 1999. The Java Virtual Machine Specification, 2o. Edición. Java Series. Addison-Wesley Prodessional.

Mace, S., Mata-Alvarez, J., 2002. Utilization of sbr technology for wastewater treatment: An overview. Ind. Eng. Chem. Res. 41, 5539–5553.

Madigan, Martinko, Parker, 1999. Brock Biología de los Microorganismos, 8o Edición. Prentice Hall Iberia, Madrid.

Marsili-Libelli, S., Vaggi, A., 1997. Estimation of respirometric activities in bioprocesses. Journal of Biotechnology 52 (3), 181–192.

Neumann, P., 2006. Communication in industrial automation - what is going on? Control Eng. Practice.

Object Management Group, 2006a. Corba/iiop specification.
URL `http://www.omg.org/technology/documents/formal/corba_iiop.htm`

Object Management Group, 2006b. Unified modelling language.
URL `http://www.uml.org/`

of the IEEE Computer Society, S. E. T. C., 1999. Ieee standard for software unit testing: An american national standard, ansi/ieee std 1008-1987. Ieee standards: Software engineering, volume two: Process standards, The Institute of Electrical and Electronics Engineers, Inc.

OSGi Alliance, 2006. Osgi service platform.
URL http://www.osgi.org

Purkhold, U., Pommering-Röser, A., Juretschko, S., Schmid, M., Koops, H.-P., Wagner, M., 2000. Phylogeny of all recognized species of ammonia oxidisers based on comparative 16s rrna and amoa sequence analysis: Implications for molecular diversity surveys. App. Env. Microbiol. 66 (12), 5368–5382.

Robert, C., Casella, G., 2004. Monte Carlo Statistical Methods, 2o. Edición. Springer-Verlag, New York, NY, USA, ISBN: 0387212396.

Rubio, M., Colomer, J., Ruiz, M. L., Colprim, J., Meléndez, J., 2004. Situation assessment in a SBR wastewater treatment process using qualitative trends. En: Vitrià, J., Radeva, P., Aguiló, I. (Eds.), FAIA-Recent Advances in Artificial Intelligence Research and Development. Vol. 113. IOS Press, Amsterdam, Netherlands, pp. 19–26, ISBN: 1-58603-466-9.

Sanz, R., C., P., W., S., De Antonio, A., 2000. Software for Complex Controllers. Springer Verlag, Cap. 7, pp. 143–164.

Schantz, R. E., Schmidt, D. C., 2001. Middleware for distributed systems: Evolving the common structure for network-centric applications. En: Encyclopedia of Software Engineering. Wiley and Sons.

Schickhuber, G., McCarthy, O., February 1997. Distributed fieldbus and control network systems. IEEE Computing and Control Engineering 8 (1), 21–32.

Söderström, T., Stoica, P., 1989. System Identification. Series in Systems and Control Engineering. Prentice Hall International, UK.

Spanjers, H., Vanrolleghem, P., Olsson, G., (eds.), P. D., 1998. Respirometry in control of the activated sludge process:principles. Rep. Tec. No.7, International Water Association.

Steinberg, D., Chesire, S., December 2005. Zero Configuration Networking, 1o. Edición. O'Reilly Media.

Strous, M., Kuenen, J., Fuerst, J. A., Wagner, M., Jetten, M. S. M., 2002. The anammox case - a new experimental manifesto for microbiological eco-physiology. Antonie van Leeuwenhoek 81, 693–702.

Tromans, D., 1999. Oxygen solubility modeling in aqueous solutions. En: Proceedings of the Second International Conference on Intelligent Processing and Manufacturing of Materials (IPMM 99). Vol. 1. pp. 411–416.

Venkatasubramanian, V., Rengaswamy, R., Yin, K., Kavuri, S. N., 2003a. A review of process fault detection and diagnosis: Part i: Quantitative model-based methods. Computers and Chemical Engineering 27, 293–311.

Venkatasubramanian, V., Rengaswamy, R., Yin, K., Kavuri, S. N., 2003b. A review of process fault detection and diagnosis: Part ii: Qualitative models and search strategies. Computers and Chemical Engineering 27, 313–326.

Venkatasubramanian, V., Rengaswamy, R., Yin, K., Kavuri, S. N., 2003c. A review of process fault detection and diagnosis: Part iii: Process history based methods. Computers and Chemical Engineering 27, 327–346.

Vinkoski, S., Jan.-Feb. 2003. Service discovery 101. IEEE Internet Computing 7 (1), 69–71.

Vogelaar, J., Klapwjik, A., van Lier, J., Rulkens, W., 1996. Temperature effects on the oxygen transfer rate between 20 and 55 c. Wat. Res. 34 (3), 1037–1041.

Wagner, M., Loy, A., Nogueira, R., Purkhold, U., Lee, N., Daims, H., 2002. Microbial community composition and function in wastewater treatment plants. Antonie van Leeuwenhoek 81, 665–680.

White, J., Speyer, J., 1987. Detection filter design: spectral theory, and algorithms. IEEE Transactions on Automatic Control 32 (7), 593–603.

WHO, 1996. Addendum - health criteria and other supporting information. En: Guidelines for Drinking-Water Quality, 2o. Edición. Vol. 2. World Health Organization, pp. 828–837.

Wimberger, D., Verde, C., July 3-8 2005. Online monitoring of an aerobic SBR process based on dissolved oxygen measurement. En: Proceedings of the 16th IFAC World Congress. Elsevier, Prague, Czech Republic, ISBN: 0-08-045108-X.

Witten, I. H., Frank, E., 2000. Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann Publishers, San Francisco, CA, USA, ISBN: 1-55860-552-5.

WP1 Process Experiments, 2004. Report on the model selection, parameter identification and validation. EOLI Deliverable D1.3, European Community Research (ICA4-CT- 2002-10012).

Yoong, E., Lant, P., Greenfield, P., 2000. In situ respirometry in an sbr treating waste water with high phenol concentrations. Wat. Res. 34 (1), 239–245.

Zimmermann, H., April 1980. Osi reference model - the iso model of architecture for open systems interconnection. IEEE Transactions on Communications 28 (4), 425–432.

Apéndice A

Introducción al lenguaje unificado de modelado (UML)

En los años 80 hubo una explosión de métodos de diseño y análisis orientado a objetos. Conforme creció la experiencia con aplicaciones, se redujo el número de métodos a unos cuantos que demostraron su utilidad mediante la aplicación exitosa en varios proyectos. Más tarde, en los años 90, la orientación a objetos ya representaba la corriente dominante en el desarrollo de software. Aparecieron también métodos refinados de segunda generación para AOO y DOO, entre estos los más aplicados eran probablemente:

- la metodología de Booch;
- la metodología OMT (Object Modelling Tool por sus siglas en inglés); y
- la metodología Fusion.

Booch y Rumbaugh (autor principal de la metodología OMT) empezaron a unir fuerzas alrededor de finales de 1994, con el objetivo de estandarizar una metodología principal para A/DOO. Dado que los dos metodologías integraron a la metodología de casos de uso de Jacobson, lo invitaron a unirse al esfuerzo en 1995.

El resultado de sus esfuerzos fue el lenguaje unificado de modelado UML (por sus siglas en ingles, Unified Modeling Language). El UML es un lenguaje estandardizado para el modelado de sistemas de software orientado a objetos, el cual permite visualizar, especificar, construir y documentar al sistema.

Dado el hecho que el UML es un lenguaje abstracto (un meta-lenguaje, el cual puede ser usado para hacer referencia a otros lenguajes, e incluso a si mismo) es aplicable para A/DOO, y para un amplio rango de otros tipos de sistemas, dominios y procesos, incluyendo los que no tienen que ver con software. En proyectos de desarrollo de software, es un estándar (Object Management Group, 2006b) que permite a capturar, comunicar y re-utilizar el conocimiento obtenido durante el desarrollo del proyecto.

Este apéndice es una introducción UML, la cual tiene como objetivo introducir los conceptos y el vocabulario básico que se utilizan en UML. La idea principal es cubrir con mla mayor brevedad posible los términos y diagramas que se utilizan en el texto de la presente tesis. Sin embargo, para un entendimiento y conocimiento más profundo se sugiere el estudio de libros de texto disponibles en calquier biblioteca que da servicio a una carrera de ciencia de computación. Como sugerencia, se recomiendan los siguientes libros:

- Fowler, M.: UML Distilled, Third Edition, Addison-Wesley, Boston, MA, 2004, ISBN: 0-321-19368-7.
- Kruchten, P.: The Rational Unified Process: An Introduction, Second Edition, Addison- Wesley, Reading, MA, 2000, ISBN: 0-201-70710-1.
- Jacobson, I., Booch,G. , and Rumbaugh, J.: The Unified Software Development Process, Addison-Wesley, Reading, MA, 1999, ISBN: 0-201-57169-2.
- Jaaksi, A., Aalto, J., Aalto, A., and Vättö, K.: Tried & True Object Development: Industry-Proven Approaches with UML, Cambridge University Press, Cambridge, 1999, ISBN: 0- 521-64530-1.
- Schneider, G., Winters, J.P.: Applying Use Cases: A Practical Guide, Addison-Wesley, Reading, MA, 1998, ISBN: 0-201-30981-5.

A.1. Resumen de diagramas básicos de UML

UML permite representar abstracciones de la estructura y del comportamiento de un sistema a traves de un modelo, el cual consiste principalmente en un conjunto de diferentes tipos de diagramas; cada uno de estos diagramas pone énfasis en aspectos y calidades particulares del sistema:

- diagramas estructurales ayudan a visualizar los elementos de la estructura del sistema, y reflejan las relaciones estáticas entre ellos. Los más utilizados son:
  1. diagramas de clases, descritos en la Sección A.2.1;
  2. diagramas de objetos, que presentan los objetos y sus relaciones en un momento del tiempo;
  3. diagramas de componentes, que representan los componentes que componen el sistema, sus relaciones e interacciones; y
  4. diagramas de despliegue, descritos en la Sección A.2.2.
- diagramas de comportamiento que ayudan a visualizar el comportamiento del sistema, y reflejan características de comportamiento y la dinámica del sistema. Los más utilizados son:
  1. diagramas de caso de uso, que muestran las relaciones entre actores y elementos del sistema, en forma de casos de uso;
  2. diagramas de actividades, que representan los procesos de alto nivel de un sistema, incluyendo el flujo de datos; también puede utilizarse para modelar lógica compleja y/o paralela dentro de un sistema;
  3. diagramas de estados, descritos en la Sección A.3.1;
  4. diagramas de secuencia, que representan una interacción, con el énfasis en la secuencia temporal de los mensajes que se intercambian; y
  5. diagramas de comunicación (en V1.0: colaboración), descritos en la Sección A.3.2.

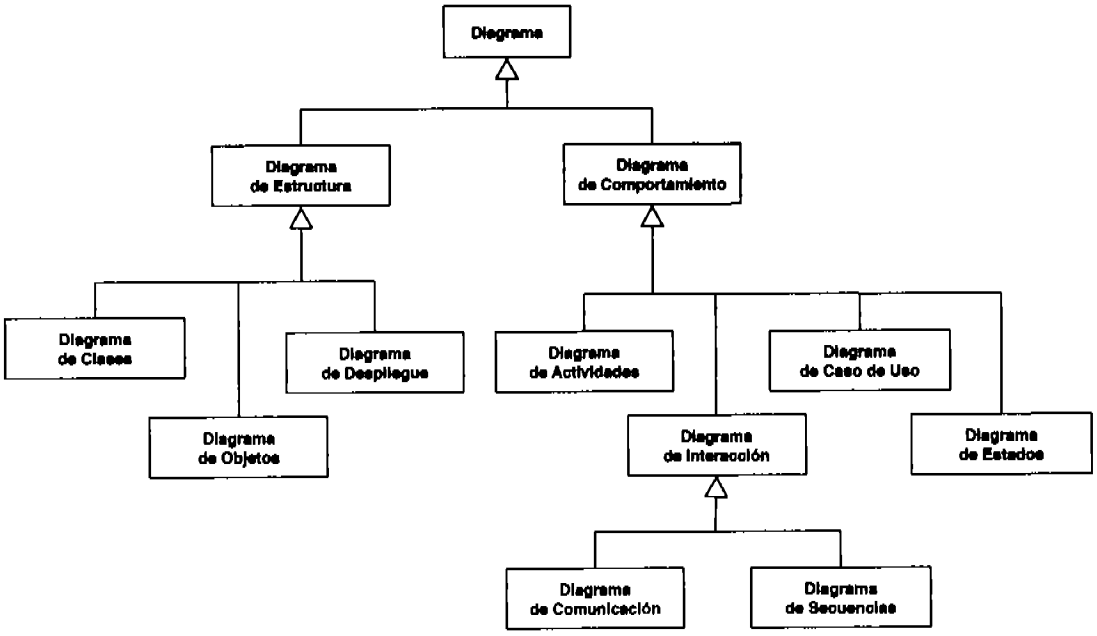


Figura A.1: Genealogía de los diagramas básicos definidos por UML; visualización de acuerdo con la notación de diagramas de clases definida por UML.

Ya que el UML es un meta-lenguaje, se puede realizar un diagrama de clases de diagramas definidos por UML, como el que se presenta en Fig. A.1, el cuál utiliza la notación visual UML de diagramas de clases y muestra la genealogía de los diagramas básicos definidos por UML.

Existen también elementos de UML que pueden integrarse en varios de los tipos de diagramas mencionados anteriormente. De estos elementos el más importante y común es la nota, la cual se presenta en un diagrama como lo muestra la Figura A.2. Una nota puede expresar una observación o descripción de partes de un diagrama al cuál se agrega.

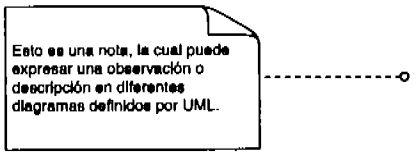


Figura A.2: Guía de la notación de notas utilizadas en diagramas de UML, de acuerdo con la definición del UML.

A continuación se presentan guías anotadas, para las notaciones de los diagramas utilizadas en la presente tesis.

A.2. Diagramas de estructura

A.2.1. Diagrama de clases

Los diagramas de clases muestran una colección de elementos de modelado declarativo (estáticos), tales como clases, tipos y, sus contenidos (e.g. atributos y métodos) y relaciones (e.g. herencia). De

acuerdo con el uso del diagrama a dentro de un proceso de desarrollo específico de software, puede reflejar diferentes niveles de detalle.

A continuación se presenta una guía de notación resumida de diagramas de clases, de acuerdo con la definición del UML (resumen de Larman, 2003), para:

1. la visualización de clases y relaciones básicas (i.e. de herencia) en la Figura A.3); y
2. la visualización de interfaces en Figura A.4.

Las dos figuras estan anotadas por notas con observaciones y descripciones de ciertos elementos.

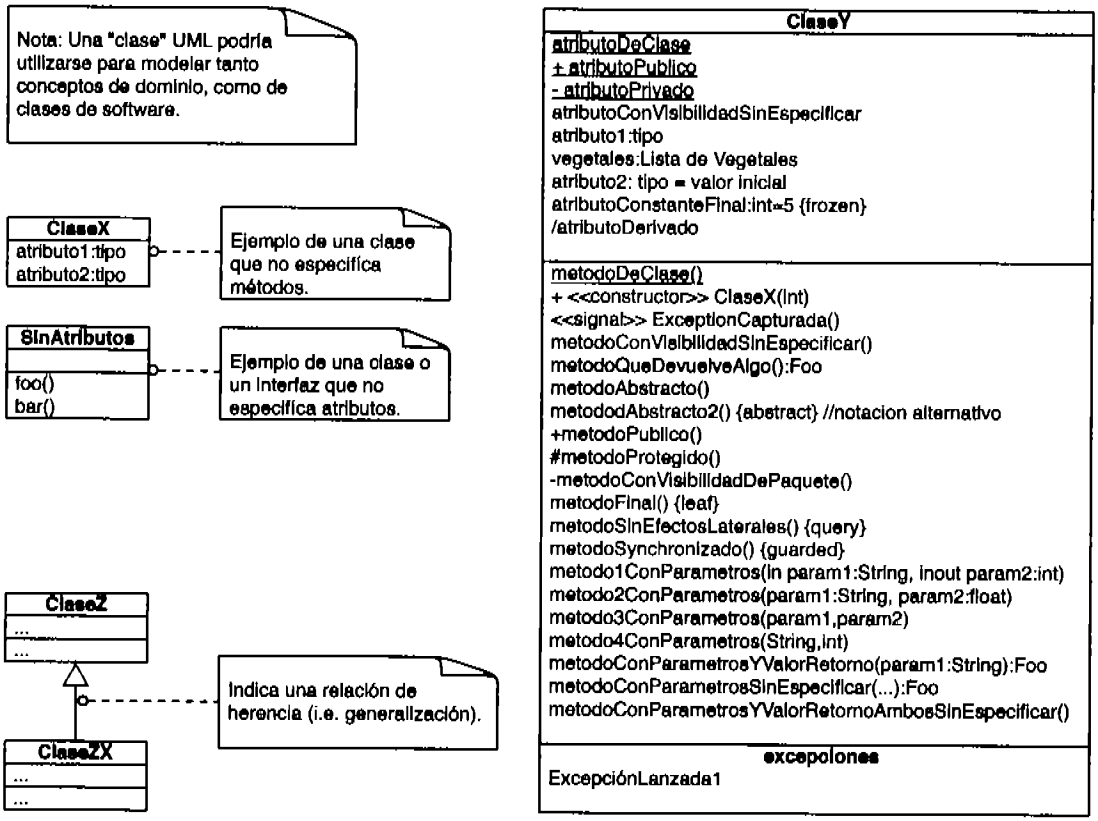


Figura A.3: Guía resumida de la notación de diagramas de clases, de acuerdo con la definición del UML (resumen de Larman, 2003), para la visualización de clases y relaciones básicas de generalización.

A.2.2. Diagrama de despliegue

Los diagramas de despliegue muestran cómo y dónde se desplegará el sistema. Las máquinas físicas y los procesadores se representan como nodos y la construcción interna puede ser representada por nodos o artefactos embebidos (i.e. nodos y componentes que en si mismo contienen nodos o componentes). La Figura A.5 presenta un guía de la notación de diagramas de despliegue, de acuerdo con la definición del UML (ejemplo de Larman, 2003).

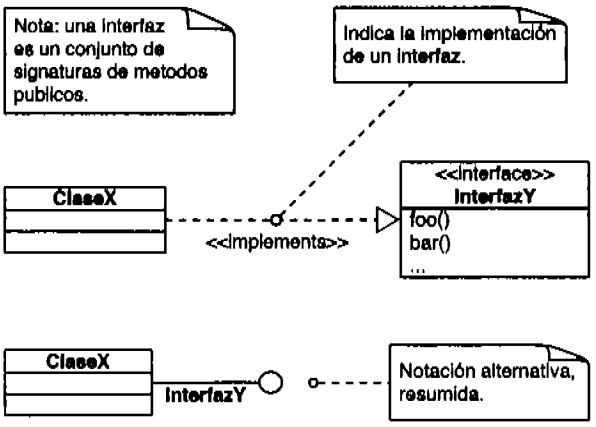


Figura A.4: Guía resumida de la notación de diagramas de clases, de acuerdo con la definición del UML (resumen de Larman, 2003), para la visualización de interfaces.

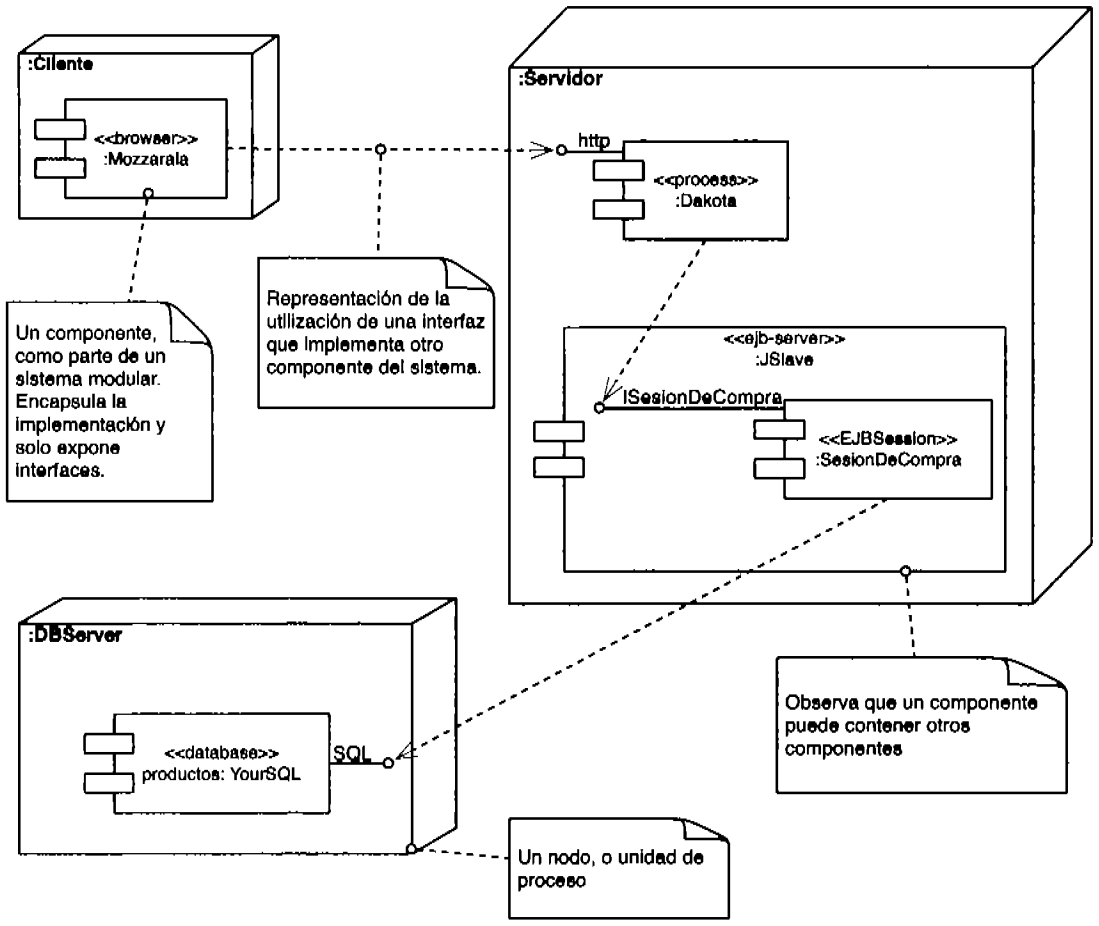


Figura A.5: Guía de la notación de diagramas de despliegue, de acuerdo con la definición del UML (ejemplo de Larman, 2003).

A.3. Diagramas de comportamiento

A.3.1. Diagrama de estados

Los diagramas de estados muestran los eventos y estados de elementos de la arquitectura (e.g. objetos, casos de uso, etc.) , y el comportamiento de los elementos como reacción a eventos. La Figura A.6 presenta una guía de la notación de diagramas de estados, de acuerdo con la definición del UML (ejemplo de Larman, 2003).

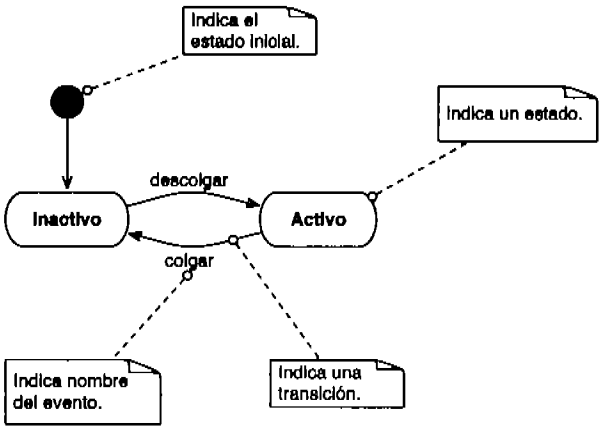


Figura A.6: Guía de la notación de diagramas de estados, de acuerdo con la definición del UML (ejemplo de Larman, 2003).

A.3.2. Diagramas de comunicación

Los diagramas de comunicación muestran la interacción de objetos, con el énfasis en la estructura interna y la comunicación mediante el intercambio de mensajes. La secuencia de los mensajes se indica mediante un esquema de numerado en una secuencia. La Figura A.7 presenta una guía de la notación de diagramas de comunicación, de acuerdo con la definición del UML (ejemplo de Larman, 2003).

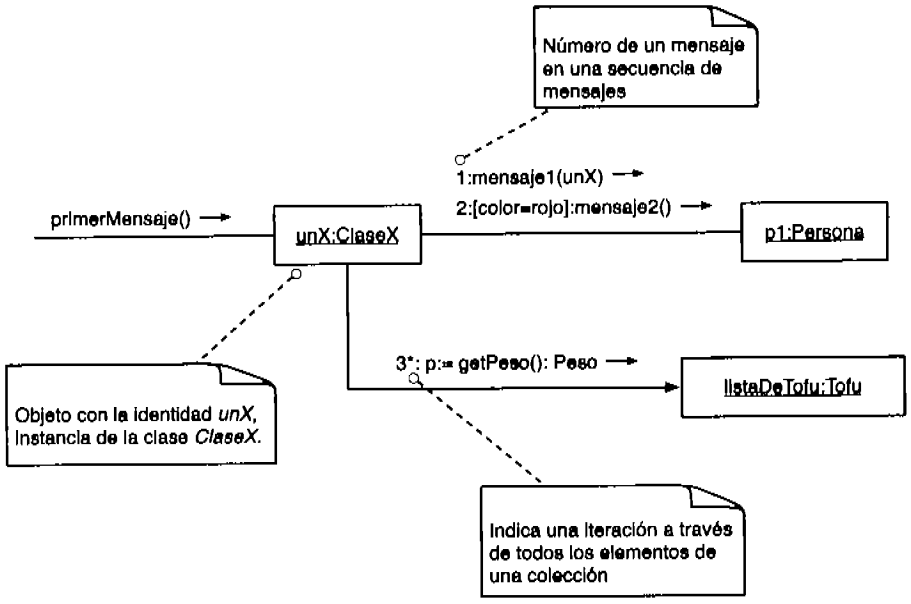


Figura A.7: Guía de la notación de diagramas de comunicación, de acuerdo con la definición del UML (ejemplo de Larman, 2003).

Apéndice B

Diseño y programación orientado a objetos (DOO, POO)

Este apéndice es una introducción a la programación orientada a objetos, la cual tiene como objetivo:

1. introducir los conceptos y el vocabulario básico que se utilizan en POO; y
2. dar una idea de como se traducen a código fuente en el lenguaje de programación Java (Gosling et al., 2005).

La idea principal es cubrir manera resumida los términos que se utilizan en la presente tesis. Sin embargo, para un entendimiento de las implementaciones se requieren conocimientos más profundos de la metodología de la programación orientada a objetos, y del lenguaje y la plataforma Java.

Para el estudio de POO y diseño orientado a objetos se recomiendan de forma general los libros de texto disponibles en cualquier biblioteca que de servicio a una carrera de ciencia de computación. Como sugerencia para el estudio de Java en particular, se recomiendan los siguientes libros:

- C. S. Horstmann y G. Cornell (traducción José Rafael García-Bermejo Giner): Core Java 2, Vol I: Fundamentos, Pearson/Prentice Hall, Madrid, España, 7a ed., 2007, ISBN: 978-84-205-4832-6.
- Bell, D. y Parr, M. (traduccion, Alfonso Vidal Romero Elizondo): Java para estudiantes, Pearson Educación, D.F., México, 3a ed., 2003, ISBN: 970-26-0144-4.

B.1. Conceptos básicos

B.1.1. ¿Qué es un objeto?

Todos los días nos rodean muchos ejemplos de objetos que forman parte de nuestro ambiente, respectivamente del mundo real: un coche, una bicicleta, un perro, un televisor, un teléfono, una pantalla, etc. Todos ellos comparten al menos las siguientes tres características; tienen:

1. un estado;
2. un comportamiento; y

3. una identidad.

Un perro tiene un estado (nombre, color, raza), un comportamiento (ladrar, vigilar) y dado que es un individuo, también tiene una identidad. Una bicicleta también tiene un estado (velocidad actual, número de velocidades), un comportamiento (acelerar, frenar, cambiar velocidad) y una identidad, ya que mi bicicleta no es la misma que la de mi amigo.

Los objetos de software modelan de manera abstracta objetos del mundo real, y por lo tanto, tienen estados, comportamientos e identidades. Un objeto de software mantiene su estado en un conjunto de *variables*, e implementa su comportamiento con *métodos*, los cuales básicamente representan funciones que pueden acceder y manipular el estado de un objeto. Su identidad es muchas veces implícita, y se representa en un programa por medio de una referencia única que determina el objeto con su conjunto independiente de variables de estado. Aparte de la posibilidad de modelar objetos del mundo real, como por ejemplo la representación de un perro como un objeto de software en un programa de animación, existe la posibilidad de la representación de conceptos abstractos, los cuales no tienen una representación física en el mundo real.

Para objetos de software, comúnmente se aplican dos principios importantes:

1. el principio de encapsulamiento, el cuál se refiere a la formación de entidades en las cuales se une un conjunto de variables con los métodos correspondientes para su manipulación; y
2. el principio de ocultación, el cual se refiere a esconder detalles internos de la implementación; en el caso de un objeto de software, exponiendo solamente un conjunto de métodos los cuales pueden ser utilizados por otras entidades.

Estos principios también suelen ser utilizados en ingeniería; un buen ejemplo es una caja de velocidades en un coche: es una entidad del coche, la cual se puede utilizar por medio de la palanca de velocidades, sin tener conocimientos detallados de como funciona todo el mecanismo internamente.

La Figura B.1 presenta de manera visual un objeto de software; se muestra en el diagrama, que el conjunto de variables forma el núcleo del objeto, y que los métodos, mostrados alrededor de este núcleo, forman una capa que esconde los detalles internos del objeto. En UML un objeto se presenta como se muestra en la Figura B.2.

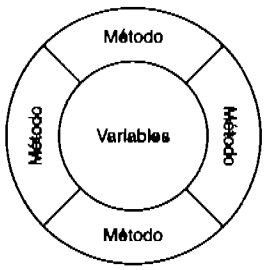


Figura B.1: Representación visual de un objeto de software, ilustrando los principios de encapsulamiento y de ocultación de información.

Se hace notar que la aplicación de estos principios permite la implementación de un sistema modular; por ejemplo, cuando falla la caja de velocidades de un coche:

1. se puede reparar o intercambiar sin que se requiera la reparación de todo el coche (i.e. es una entidad encapsulada); y

2. siempre y cuando las interfaces entre la caja de velocidades y el coche sean compatibles, se puede reemplazar por otra caja de velocidades que internamente no funciona exactamente del mismo modo que la anterior (i.e. ocultación de los detalles al interior de la entidad).

Estas posibilidades también son interesantes para sistemas de software, tanto para manejar la complejidad, como para la construcción de sistemas que requieren adaptación o cambios a lo largo de su ciclo de vida.

Figura B.2: Representación visual de un objeto, de acuerdo con la notación de diagramas de comunicación de UML.

B.1.2. ¿Qué es un mensaje?

Una aplicación de software en POO usualmente consiste en un conjunto de diferentes objetos, y proporciona funcionalidad y un comportamiento complejo por medio de la interacción de estos objetos. Está interacción se realiza por medio del intercambio de mensajes: cuando el objeto A quiere ejecutar un método del objeto B, el objeto A manda un mensaje al objeto B; si se requiere información adicional para la ejecución de un método, esta información se puede pasar en forma de parámetros. Por lo tanto un mensaje se define por tres elementos:

1. el objeto al cual se emite el mensaje;
2. el método que se pretende ejecutar; y
3. los parámetros requeridos por este método.

La Figura B.3 presenta un mensaje intercambiado entre dos objetos, de acuerdo con la notación de diagramas de comunicación de UML.

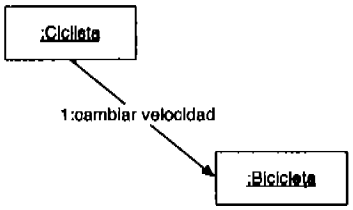


Figura B.3: Representación visual de un mensaje entre dos objetos, de acuerdo con la notación de diagramas de comunicación de UML.

B.1.3. ¿Qué es una clase?

En el mundo real siempre existen muchos objetos del mismo tipo. Por ejemplo, una bicicleta específica es solamente una de muchas en el mundo. Usando terminología de POO, está bicicleta específica

es una instancia de una clase de objetos conocidos como bicicletas, las cuales comparten ciertas características en forma de posibles estados y comportamientos; sin embargo, el estado de cada bicicleta es independiente y puede ser diferente al estado de otras bicicletas.

Cuando se construye una bicicleta, se puede tomar ventaja de las características compartidas, usando un plano que define las características compartidas, para construir muchas bicicletas del mismo modelo. En realidad, seria muy poco eficiente una producción que utiliza un nuevo plano para cada bicicleta que se produce.

En un proyecto de software desarrollado de acuerdo con la metodología de DOO, es posible tener muchos objetos de software que comparten características. Como en el caso de las bicicletas, es posible construir un plano para estos objetos, los cuales se denominan *clase*. Una clase se define por:

1. un nombre;
2. un conjunto de atributos; y
3. un conjunto de métodos que representan posibles comportamientos.

Una clase es un plano que permite la creación de un cierto tipo de objetos, los cuales puedan ser utilizados en una aplicación de software. Los objetos creados a partir de una clase se denominan *instancias* de la clase; por lo tanto y en el contexto de POO cuando se habla de:

1. una *instancia*, se refiere a un objeto de un cierto tipo, definido por una clase específica;
2. un *tipo*, se refiere a una clase específica que define una variedad de objetos; y

Las instancias de una clase comparten los métodos definidos por la clase, pero no comparten las variables que representan los atributos que defina la clase. Estas variables se denominan *variables de instancia*, y representan el estado individual de cada instancia.

Para clases del mundo real es obvio que no son los objetos que describen: el plano de una bicicleta no es una bicicleta. En software no siempre es tan fácil distinguir entre clases y objetos, en parte, porque de antemano el software representa un modelo abstracto del mundo real o de conceptos ya abstractos. Sin embargo, una diferencia importante es que las clases, en comparación con los objetos que proporcionan el beneficio de modularidad, proporcionan el beneficio de la reutilización.

La Figura B.4 presenta los principios de un modelo de una bicicleta, en forma de una clase, y de acuerdo con la notación de diagramas de clases de UML.

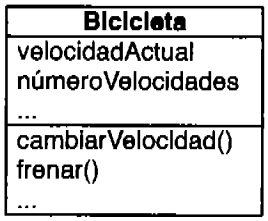


Figura B.4: Representación visual parcial de la clase Bicicleta, de acuerdo con la notación de diagramas de clases de UML.

B.1.4. ¿Qué es la herencia?

Los objetos se definen por medio de clases, y la clase contiene la información sobre los atributos y el posible comportamiento de un objeto. Esto se refleja en el mundo real; e.g. si uno sabe que un "AMP 2007 Cross Country XT" es una bicicleta, también puede fácilmente saber que por lo menos tiene dos ruedas, dos pedales y un manubrio. El mecanismo para obtener clases a partir de objetos del mundo real es usualmente la abstracción; este mecanismo también se puede aplicar para definir clases a partir de otras clases, con diferentes niveles de abstracción: por ejemplo, bicicletas de montaña, bicicletas de carreras y bicicletas dobles son diferentes variedades de bicicletas.

Bicicletas de montaña, bicicletas de carreras y bicicletas dobles son todas bicicletas, que comparten las características de una bicicleta. La terminología en POO es la siguiente:

1. la clase bicicleta es *superclase* de bicicletas de montaña, bicicletas de carreras y bicicletas dobles;
2. las clases de bicicletas de montaña, bicicletas de carreras y bicicletas dobles son *subclases* de la clase bicicleta;
3. las subclases heredan los atributos y métodos de la superclase;
4. los atributos y métodos heredados se denominan la *herencia.*

La Figura B.5 muestra como se visualiza la herencia en un diagrama de clases, de acuerdo con la notación definido por el UML.

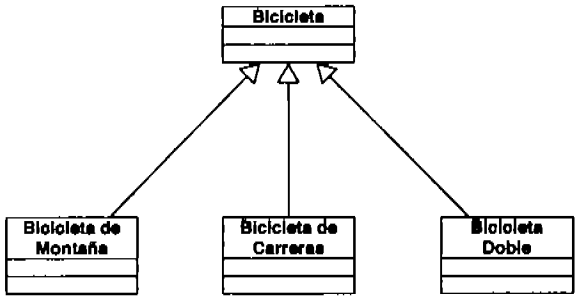


Figura B.5: Representación visual de herencia, de acuerdo con la notación de diagramas de clases de UML.

En lugar de estar limitados al estado y comportamiento heredado, las subclases pueden:

1. añadir nuevas definiciones de variables y de métodos aparte de los atributos y métodos que heredaron de su superclase; y
2. modificar un comportamiento heredado.

Se hace notar que en POO esto último representa un concepto que se llama *polimorfismo*: dos o más clases de objetos pueden responder al mismo mensaje de formas diferentes, utilizando operaciones polimorfas.

El concepto de herencia en POO no está limitado a un solo nivel; existe la posibilidad de crear una jerarquía con el número de niveles necesarios, la cual representa un árbol de herencia. Los métodos y atributos se heredan hacia los niveles que están más abajo, y generalmente el nivel en la jerarquía corresponde al nivel de abstracción:

- superclases representan conceptos más generales; y
- subclases representan conceptos más especializados.

Esta característica de POO actualmente proporciona el mecanismo para manejar la complejidad del software para una aplicación, sin embargo, requiere de experiencia y tiempo para aprender a diseñar jerarquías con generalizaciones y especializaciones claras y adecuadas para el problema que se intenta resolver.

B.1.5. ¿Qué es una interfaz?

En el mundo real una interfaz es una conexión física y funcional entre dos aparatos o sistemas independientes. De acuerdo con esta definición, por ejemplo un control remoto representa una interfaz entre nosotros y un televisor, la cual nos permite cambiar canales, el volumen, etc.

En POO existe el concepto de una interfaz de manera que permite la interacción entre dos objetos independientes y no relacionados en una jerarquía de clases. Una interfaz:

1. representa un contrato de comportamiento, lo cual es como un protocolo que define todos los mensajes posibles que se pueden intercambiar con un objeto cuya clase implementa una cierta interfaz;
2. puede ser implementado por cualquier clase en la jerarquía de clases; y
3. hace utilizable similitudes entre clases que no están relacionados en la jerarquía, sin forzar artificialmente una relación de herencia.

Por ejemplo: una bicicleta en una tienda es al mismo tiempo un artículo del inventario de esta tienda, con un precio, un número que identifica al artículo etc. Ser un artículo es un concepto diferente e independiente del hecho de ser una bicicleta. Entonces, la definición de una interfaz para representar un artículo, permite a una clase bicicleta implementar esta interfaz, y consecuentemente, comportarse como un artículo.

La Figura B.6 presenta las dos posibles visualizaciones de una clase Bicicleta que implementa un interfaz Artículo, de acuerdo con la notación de diagramas de clases de UML.

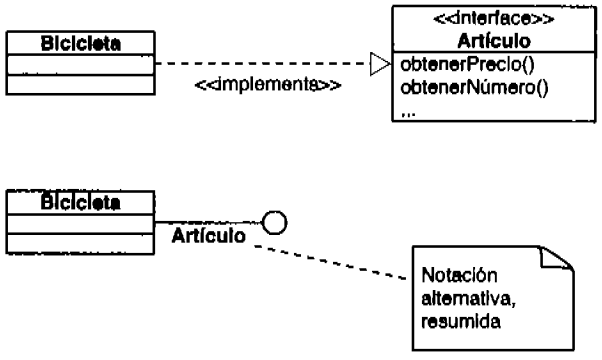


Figura B.6: Representaciones visuales de la implementación de una interfaz, de acuerdo con la notación de diagramas de clases de UML.

B.2. Conceptos básicos traducidos a Java

Java es un lenguaje de programación que permite el diseño de aplicaciones dentro del marco de referencia de la programación orientada a objetos (Gosling et al., 2005). Dado el hecho que ha sido inventado específicamente para POO, Java permite la expresión directa de conceptos de POO por medio de la gramática del lenguaje. Aparte de elementos gramaticales para clases e interfaces, existen elementos gramaticales para la aplicación de:

1. el principio de encapsulamiento;
   Java permite la aplicación del principio de encapsulamiento en dos niveles:

   a) al nivel del elemento gramatical de una clase; y

   b) al nivel de conjuntos de clases, los cuales se denominan *paquetes;*

2. el principio de la ocultación de información;
   Java permite la especificación de la visibilidad y accesibilidad de atributos y métodos, por medio de las siguientes etiquetas:

   a) público (`public`), la cual define una visibilidad y accesibilidad pública;

   b) protegido (`protected`), la cual define una restricción de la visibilidad y accesibilidad a subclases o clases del mismo paquete; y

   c) privado (`private`), la cual define la ocultación completa;

B.2.1. Interfaces

La interfaz que se presentó en la Figura B.6, se puede presentar en UML con más detalle, mostrado en la Fig. B.7, y, traducir directamente al siguiente código fuente en Java:

```
public interface Artículo {
  public float obtenerPrecio();
  public long obtenerNúmero();
}//interface Artículo
```

Se hace notar que una interfaz:

1. tiene solamente la declaración de los métodos que forman parte del contrato de comportamiento; y

2. que los métodos son públicos (`public`) por omisión.

B.2.2. Clases

La clase bicicleta que se presentó anteriormente (Fig. B.4), se puede representar en UML con más detalle (Fig. B.8) y, traducir directamente al siguiente código fuente en Java:

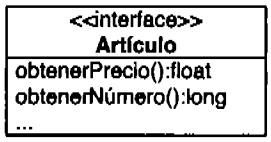


Figura B.7: Representación visual de la interfaz "Artículo" con más detalles (... significa que no es un ejemplo completo), de acuerdo con la notación de diagramas de clases de UML.

```
public class Bicicleta {
  //atributos
  private int velocidadActual;
  private int numeroDeVelocidades;
  //.. otros atributos
  //constructor para nuevas instancias
  public Bicicleta() {
    [...]
  }//constructor

  //métodos
  public void cambiarVelocidad(int velocidad) {
    //... implementación del comportamiento
  }//cambiaVelocidad

  public void frenar() {
    //... implementación del comportamiento
  }//frenar

  //... otros métodos
}//clase Bicicleta
```

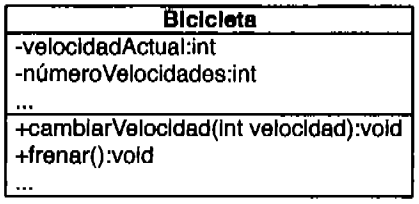


Figura B.8: Representación visual de la clase bicicleta con más detalles (... significa que no es un ejemplo completo), de acuerdo con la notación de diagramas de clases de UML. Las etiquetas de visibilidad/accesibilidad que se usan en la notación son: "-" para privado, "#" para protegido y "+" para público.

B.2.3. Objetos

Aparte de la declaración de los atributos y la implementación de los métodos de la clase, cada clase debe contar con un método especial, que permite la construcción de nuevos objetos que son

instancias de esta clase, y que se denomina *constructor*:

```
public Bicicleta() {
  [...]
}//constructor
```

Para crear una nueva instancia en Java, se puede utilizar la siguiente expresión:

```
[...]
Bicicleta miBicicleta = new Bicicleta();
[...]
```

Se hace notar que existe la posibilidad de implementar varios constructores con diferentes conjuntos de parámetros, para crear instancias con diferentes estados iniciales.

B.2.4. Mensajes

El intercambio de mensajes entre dos objetos en el mismo contexto de ejecución (i.e. en este caso una máquina virtual de Java), es la invocación de un método de un objeto, por medio de otro objeto. En el ejemplo mostrado en la Figura B.9, una instancia de la clase `Ciclista` emite el mensaje `cambiarVelocidad` a un objeto de la clase `Bicicleta` por medio de la referencia `miBicicleta`.

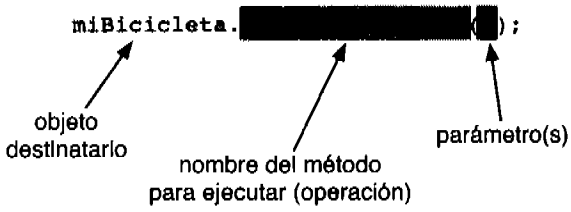


Figura B.9: Mensaje en Java con anotaciones que identifican los elementos importantes del mensaje.

Apéndice C

El Modelo ISO/OSI

Este apéndice es una introducción resumida del modelo ISO/OSI (Zimmermann, 1980) para el diseño y la descripción de protocolos de comunicación en redes de computadoras. Tiene como objetivo:

1. introducir el concepto básico de la comunicación bajo las reglas del modelo;
2. presentar el vocabulario básico que se utiliza en el modelo; y
3. introducir las capas y su funcionalidad correspondiente como lo define el modelo.

C.1. Mecanismo de la comunicación

Para enviar datos de una aplicación en un nodo A (el transmisor) a otra aplicación en un nodo B (el receptor), se requiere un mecanismo de comunicación que realice el transporte de los datos. El estándar ISO/OSI (Zimmermann, 1980), es un modelo abstracto de referencia, para el diseño y la descripción de protocolos de comunicación en redes de computadoras. Este modelo define un mecanismo de comunicación para el transporte de datos, en el cual los datos pasan a través de una pila de capas que realizan funciones independientes, con el objetivo de que en conjunto puedan realizar un transporte físico de los datos entre sistemas heterogéneos.

La Figura C.1 presenta una visualización general de este mecanismo de comunicación.

La idea principal de lo que es una pila de capas en el modelo ISO/OSI es obtener la mayor independencia posible entre las diferentes capas, de manera que:

1. permita cambiar capas inferiores con diferentes implementaciones sin afectar las capas superiores; y
2. permita ensamblar una pila adecuada para formar un protocolo específico que cumpla con ciertos requisitos anteriormente definidos.

Para que esto funcione, se requiere que:

1. cada capa en el transmisor tenga una capa correspondiente en el receptor;
2. cada capa pueda comunicar con la capa correspondiente en el receptor.

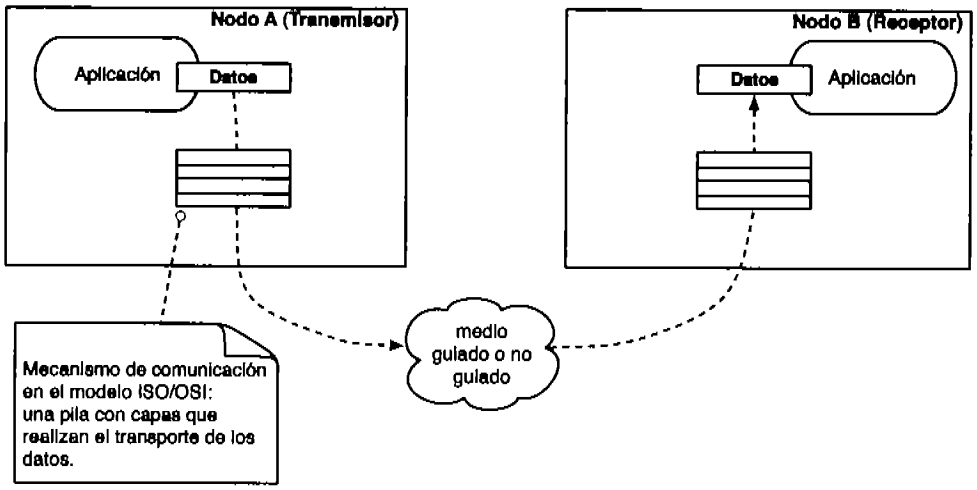


Figura C.1: Comunicación entre aplicaciones en diferentes nodos, a través de la implementación de la pila de capas del modelo ISO/OSI.

La comunicación entre capas correspondientes en el transmisor y el receptor funciona por medio de información añadida en la capa y este mecanismo se denomina encapsulamiento (distinto al mecanismo llamado encapsulamiento en POO): se encapsula la información recibida de la capa superior en uno o múltiples paquetes que contienen información adicional al principio, denominada cabeza, y/o al final, denominada cola. El proceso del encapsulamiento, mostrado en la Figura C.2, ocurre en el momento de la transmisión, y se invierte en el momento de la recepción.

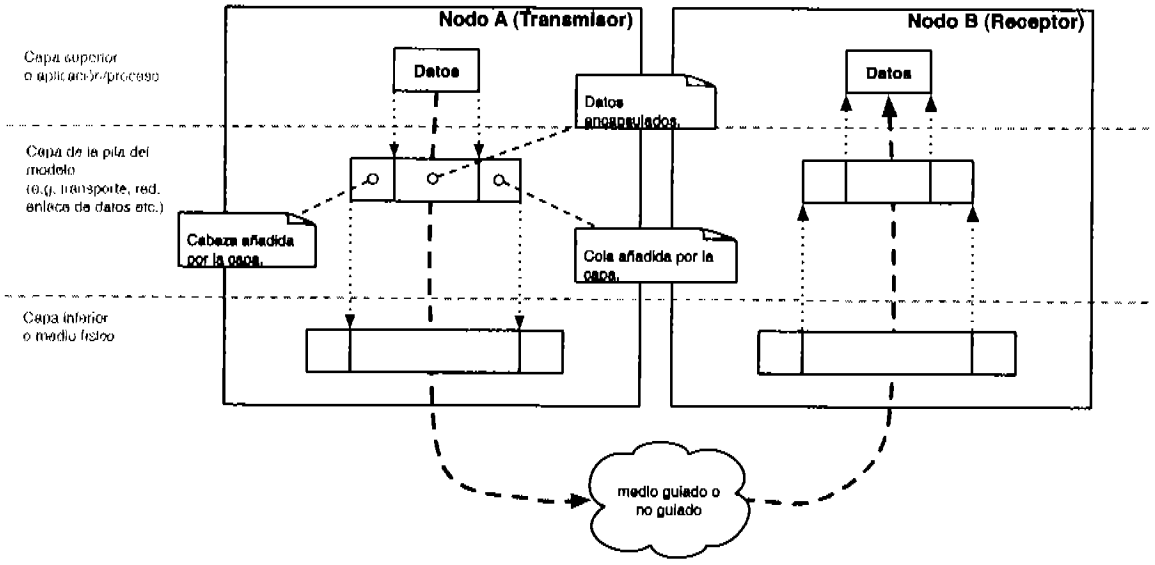


Figura C.2: Proceso de encapsulamiento de datos a través de la pila de capas del modelo ISO/OSI .

En algunos casos una capa inferior puede tener límites en cuanto al tamaño de los paquetes que se pueden transmitir. Este tipo de límite comúnmente se denomina MTU (por sus siglas en ingles: maximum transport unit): es la entidad de tamaño máximo que se puede transportar. En estos casos una capa de la pila puede:

- en el transmisor: cortar un paquete de la capa superior, encapsular y transmitir las partes cortadas en múltiples paquetes, utilizando los servicios de su capa inferior o al medio de transmisión; éste proceso se denomina fragmentación; y
- en el receptor: reunir múltiples paquetes de la capa inferior, utilizando información de la secuencia de cada fragmento encapsulado (usualmente se encuentra en la cabeza añadida por la capa responsable para la fragmentación en el transmisor), y pasando la información en un solo paquete a la capa superior; este proceso se denomina unificación.

Los dos procesos anteriores se muestran en la Figura C.3. Se hace notar, que en algunos casos es deseable evitar la fragmentación y unificación en la pila de capas, por lo cual se define el tamaño máximo de un paquete de datos por medio del tamaño máximo que se puede transmitir por la capa física, substrayendo el tamaño de la información añadida en el proceso de encapsulamiento en forma de cabezas y colas, que usualmente son de tamaño fijo.

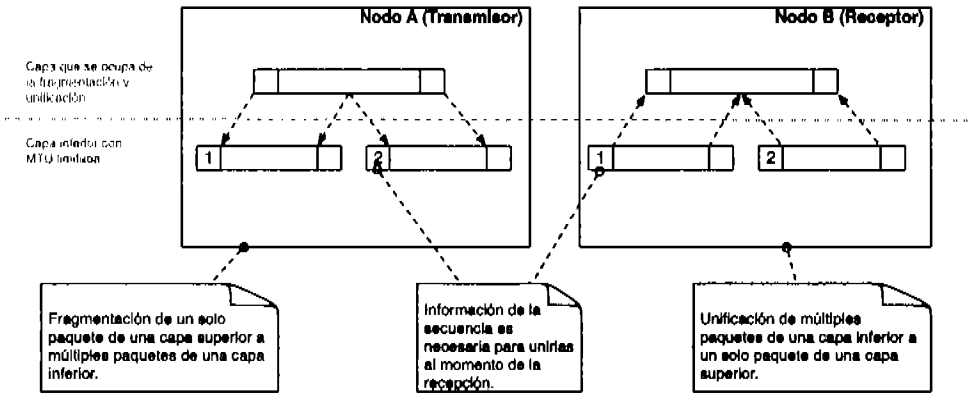


Figura C.3: Proceso de fragmentación y unificación de paquetes en el proceso de encapsulamiento a través de las capas del modelo

C.2. Las capas del modelo ISO/OSI

El modelo ISO/OSI consiste en una pila de siete capas, presentada en la Figura C.4, en la cual cada capa tiene una funcionalidad específica y definida que:

1. puede dar un servicio a la capa superior; y
2. utiliza los servicios de la capa inferior.

A continuación se presenta un resumen de la funcionalidad de cada una de estas capas definidas por el modelo.

C.2.1. La capa física

La capa física del modelo maneja la comunicación en una red al nivel del medio de transmisión, el cual constituye el soporte físico, a través del cual emisores y receptores pueden comunicarse. Existen dos tipos de medios:

Figura C.4: Las siete capas del modelo ISO/OSI

1. guiados, donde ondas son conducidas a través de un camino físico (e.g. cable coaxial, cables de pares, fibra óptica); y
2. no guiados, donde el medio solo proporciona un soporte para la transmisión de las ondas, pero no las guía (e.g. aire, vacío).

Relacionado con estos medios, hay diferentes formas en las que se puede transmitir la información representada por bits, los cuales deben ser parte de una especificación de la capa física (i.e. codificación de señal, niveles de tensión/intensidad de corriente eléctrica, modulación, tasa binaria, etc.).

C.2.2. La capa de enlace de datos

Esta capa debe manejar la transferencia fiable de datos entre entidades de una red local, detectando y posiblemente corrigiendo errores de transmisión que pueden ocurrir en la capa física. Por lo tanto, la capa de enlace de datos se ocupa de:

- el direccionamiento físico de las entidades;
- la topología de la red;
- el acceso a la red; y
- la notificación de errores.

Además puede contar con algún mecanismo de regulación del tráfico que evite la saturación de un receptor que sea más lento que el emisor.

C.2.3. La capa de red

La capa de red se ocupa de la transferencia de datos de una longitud variable entre entidades de una red, aún cuando ambos no estén conectados directamente (i.e. subredes o redes interconectados). La tarea de está capa incluye:

- dirigir paquetes de datos;
- la fragmentación y unificación de paquetes para la adaptación a las posibilidades en la capa de enlace de datos;

- la gestión de la congestión de la red, un fenómeno que se produce cuando la saturación en un nodo afecta a toda la red; y
- mantener una cierta calidad de servicio requerida por la capa de transporte.

Comúnmente existen dispositivos especiales que facilitan ésta tarea, denominados encaminadores (aunque es más frecuente encontrar el nombre en inglés: routers).

C.2.4. La capa de transporte

Está capa proporciona la transferencia de datos de manera confiable a las capas superiores. Está tarea incluye:

- mantener un vinculo de comunicación entre nodos de la red;
- controlar el flujo de datos entre nodos de la red;
- detección de errores en la transmisión; y
- en algunos casos, la retransmisión de paquetes erróneos.

Además, a partir de ésta capa, el servicio que se presta a las capas superiores ya no debe tener dependencias y limitaciones relacionadas con el tipo de la red física que se esté utilizando.

C.2.5. La capa de sesión

La capa de sesión se ocupa de las conexiones o diálogos entre aplicaciones o procesos en nodos:

- establece, gestiona y termina la sesión entre emisor y receptor;
- controla quién transmite y quién escucha, evitando concurrencia en operaciones críticas;
- mantiene puntos de verificación (checkpoints), que sirven para que ante una interrupción de la transmisión, se pueda reanudar la transmisión y efectuar todas las operaciones definidas de principio a fin.

Se hace notar que, en muchos casos, los servicios de la capa de sesión son parcialmente, o incluso, totalmente prescindibles.

C.2.6. La capa de presentación

El objetivo de la capa de presentación es la gestión de la presentación de la información. Establece un contexto entre entidades que utilizan diferente semántica o sintaxis de los datos transmitidos, traduciéndolos a una presentación adecuada para entidades en capas superiores, siempre y cuando exista una traducción posible. Con este fin, se estableció una estructura para la presentación, utilizando reglas de la notación abstracta de sintaxis uno ASN.1 (por sus siglas en ingles, Abstract Syntax Notation One).

En comparación a las capas inferiores, ésta se encarga principalmente de los contenidos de los paquetes.

C.2.7. La capa de aplicación

Esta capa es la interfaz para aplicaciones y procesos, la cual proporciona servicios que requieren comunicación por medio de una red, utilizado la capa de presentación. Se hace notar que está capa proporciona el servicio a los procesos y aplicaciones, no al usuario final. Ejemplos de protocolos correspondientes a ésta capa son:

- SMTP (por sus siglas en ingles, Simple Mail Transfer Protocol);
- FTP (por sus siglas en ingles, File Transfer Protocol); y
- HTTP (por sus siglas en ingles, Hypertext Transfer Protocol).

Apéndice D

Pruebas de entidades

Las pruebas de entidades se diseñan para comprobar el funcionamiento de implementaciones de entidades como clases o módulos de software de acuerdo con su especificación. La aplicación de una prueba a una implementación específica puede revelar errores, los cuales se definen como desviaciones del funcionamiento especificado. El objetivo de las pruebas de entidad es encontrar errores antes de usar las entidades y de esta manera mejorar la calidad del software que se entrega para el uso. Con este objetivo, han sido establecido como estándar para proyectos de software por el IEEE (of the IEEE Computer Society, 1999).

En este trabajo, los entidades de prueba son clases y el diseño de las pruebas está basada en la biblioteca jUnit (Gamma y Beck, 2000). Las pruebas se aplican de manera automatizada durante el proceso de compilación y generación de los componentes.

Por ejemplo: dado una clase que específica un contador con un método para incrementar el propio valor por uno,

```
/**
 * Increments this <tt>Counter</tt> by one.
 *
 * @return the resulting counter value.
 */
public int increment();
```

se puede diseñar una prueba de la siguiente manera:

```
public void setUp() throws Exception {
  super.setUp();
  m_Counter = new Counter(0);
}//setUp

public void testIncrement() throws Exception {
  assertEquals(1,m_Counter.increment());
  assertEquals(2,m_Counter.increment());
  assertEquals(3,m_Counter.increment());
}//testIncrement
```

Apéndice E

Ambiente de pruebas

E.1. Hardware

El sistema del ambiente de pruebe consiste en la siguiente hardware:

1. nodo 1, una computadora portatil de Toshiba, Satelite 1730, Intel Celeron 699 MhZ, con 128Mb RAM;
2. nodo 2, una computadora portatil Apple, PPC, 1.67 Ghz, con 1Gb RAM;
3. nodo 3, una computadora Dell, PowerEdge 600SC, Intel Pentium 4, 2.4 Ghz, con 512MB RAM;
4. switch, de Linksys, 100 mbit Fast Ethernet, full duplex (100BASE-TX); y
5. cables (Ethernet, CAT5) necesarios para las conexiones.

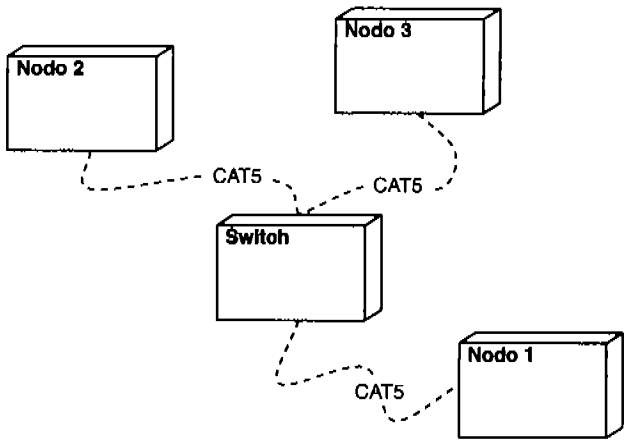


Figura E.1: Diagrama de nodos (UML) del ambiente de pruebas

E.2. Software

Los nodos del sistema del ambiente de prueba operaron con:

1. Linux, RedHat 9, versión del Kernel 2.4.20, "1.5.0" Java(TM) 2 Runtime Environment, Standard Edition Java HotSpot(TM) Client VM (build 1.5.0, mixed mode, sharing).
2. Linux, RedHat 7.2, versión del Kernel 2.4.7-10, "1.5.0" Java(TM) 2 Runtime Environment, Standard Edition Java HotSpot(TM) Client VM (build 1.5.0, mixed mode, sharing).
3. Mac OS X, 10.4 en modo Unix, y con "1.5.0_07" Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_07-164) Java HotSpot(TM) Client VM (build 1.5.0_07-87, mixed mode, sharing)

Todos los nodos utilizaron el contenedor OSGi Equinox en la versión 3.2, compatible con la especificación OSGi R4, dado que representa la implementación de referencia para la especificación. Para la configuración Zeroconf IPv4 (Link-Local) en las maquinas con Linux se utilizó el paquete denominado *zcip*.

Apéndice F

Glosario

abstracción el acto de reunir las cualidades esenciales o generales de cosas similares.

aislamiento de fallas resolución de tipo, ubicación y tiempo de la detección de una falla. Comúnmente es el paso siguiente después de la detección de una falla (Isermann y Ballé, 1997).

algoritmo conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.

análisis orientado_a_objetos Estudio del campo de un problema o sistema en función de conceptos del campo, como clases conceptuales, asociaciones y cambios de estado.

AOO → *análisis orientado a objetos*

API de sus siglas en inglés Application Programming Interface. En el contexto de orientación a objetos es un conjunto de clases, y interfaces que se pueden utilizar para diseñar un componente de una aplicación.

atributo una característica o propiedad de una clase a la que se le asigna un nombre.

arquitectura estructura lógica y física de los componentes (→*módulo*) de un sistema, por ejemplo de →*software.*

biomasa → *lodos activos*

bulto → *componente informático* en el contexto de OSGi. Representa una entidad que se puede instalar e ejecutar en un contenedor compatible con la especificación OSGi.

ciclo de vida un conjunto de fases sucesivas en un proyecto. En el caso de una planta, usualmente la primera fase del proyecto es la planificación, y luego siguen fases como contrucción, validación y operación. El termino ciclo se refiere al hecho que finalmente calquier proyecto llega a la fase que marca el final de su vida, por ejemplo cuando el avance tecnológico y cambios economicos requieren que se cierra o reemplaza a la planta .

componente informático es un →*módulo* de →*software*; contiene el código binario ejecutable y datos que representan recursos requeridos para la ejecucción del codigo binario (e.g. de configuración).

CORBA de sus siglas en inglés Common Object Request Broker. → *middleware*

CSMA/CD de sus siglas en inglés Carrier Sense Multiple Access/Collision Detection. Es una técnica utilizada en dispositivos → *Ethernet* para manejar el acceso al medio físico. Se puede detectar una colisión, sin embargo, después de una colisión los nodos esperan un tiempo aleatorio antes de volver a intentar transmitir.

DAF Detección y aislamiento de fallas

DES sistema de eventos discretos, por sus siglas en inglés Discrete Event System (Cassandras, 1993).

descriptor adj. que describe.

detección de fallas resolución de fallas presentes en un sistema y el tiempo de la detección (Isermann y Ballé, 1997).

diagnóstico de fallas resolución de tipo, tamaño, ubicación y tiempo de detección de una falla. Comúnmente es el paso siguiente, después de la detección de una falla e incluye el aislamiento y la identificación (Isermann y Ballé, 1997).

diseño orientado a objetos Un Proceso que utiliza los productos del → *análisis orientado a objetos* para la especificación de un sistema en función de objetos, clases, interfaces y sus interacciones.

DOO → *diseño orientado a objetos*

eficacia Capacidad de lograr el efecto que se desea o se espera.

eficiencia Capacidad de lograr el efecto que se desea o se espera y al mismo tiempo minimizando el esfuerzo y uso de recursos.

emparejamiento Acción y efecto de emparejar.

emparejamiento máximo Emparejamiento con el número máximo posible de parejas de vertices.

emparejar Juntar vértices formando parejas en una gráfica bipartida.

encapsulamiento Mecanismo que se utiliza para ocultar los datos, la estructura interna y los detalles de implementación de algunos elementos, como un objetivo o un subsistema. Todas las interacciones con un objeto se realizan a través de una interfaz pública de operaciones.

enlace Unión, conexión de algo con otra cosa.

espacio de tuplas es conceptualmente una memoria compartida que permite añadir, remover y actualizar datos en forma de tuplas (→ *tupla*) (Gelernter, 1992).

ethernet nombre de una tecnología de redes de área local, que se refiere a una red de área local bajo el estándar IEEE 802.3.

características extrínsecas características externas, que un objeto o sistema tiene en relación a otros objetos y/o el ambiente.

falla desviación no permitida de al menos una propiedad característica o de un parámetro del sistema de su condición nominal (Isermann y Ballé, 1997).

FSM máquina de estados finitos, de sus siglas en inglés Finite State Machine.

identificación de fallas resolución de tamaño y comportamiento temporal de una falla. Comúnmente es el paso siguiente después del aislamiento de una falla (Isermann y Ballé, 1997).

IMM interfaz máquina-máquina; módulo de software, llamado la interfaz-máquina-máquina (IMM), el cual puede mediar entre otros módulos de software del SAOP y módulos de hardware que tienen enlaces con el nivel de campo.

Interfaz Un conjunto de signaturas de operaciones públicas.

IP el protocolo de Internet (IP, de sus siglas en inglés: Internet Protocol) para la comunicación entre nodos físicamente distribuidos a través de una red heterogénea (Internet).

ISO organización internacional de estándares, de sus siglas en inglés International Standards Organization.

lenguaje de programación orientado a objetos Un lenguaje de programación que soporta los conceptos de encapsulación, herencia, y polimorfismo.

lodos activos microorganismos uni- y multicelulares en suspensión, utilizados para el tratamiento biológico de aguas residuales. Principalmente son bacterias y hongos que forman comunidades que se adaptan a las condiciones en la planta.

memoria volátil memoria cuya información se pierde al interrumpirse el flujo de corriente eléctrica.

middleware software que tiene una parte central en la arquitectura de un sistema distribuido, muy parecido al patrón de diseño *mediador* (Gamma et al., 1995). Maneja los detalles de la comunicación por medio de una red, presentando una interfaz sencilla al programador de un componente distribuido. Definición por Schantz y Schmidt (2001).

mnemónica una etiqueta fácil de memorizar.

modular perteneciente o relativo al →*módulo*.

módulo Pieza o conjunto unitario de piezas que se repiten en una construcción de cualquier tipo, para hacerla más fácil, regular y económica.

monitoreo tarea contínua ejecutada en tiempo real para determinar las condiciones de un sistema físico mediante la grabación de información y, el reconocimiento y la indicación de condiciones anormales en el comportamiento del sistema (Isermann y Ballé, 1997).

MSE promedio de los errores cuadrados, de sus siglas en inglés Mean Squared Error.

$$MSE = \frac{1}{n}\sum_{1}^{n}(\theta_1 - \theta_2)^2$$

multicast envio de un mensaje de un nodo a varios otros nodos al mismo tiempo por medio de una red.

NS servicio en una red de área local (NS, de sus siglas en inglés: Network Service).

NSD el descubrimiento de servicios de red en una red de área local (NSD, de sus siglas en inglés: Network Service Discovery). Permite la detección de direcciones y puertas de servicios para establecer una conexión y consumir el servicio. Se puede utilizar para la auto-configuración de un sistema distribuido.

OSGi de sus siglas en inglés Open Service Gateway interface. Es una especificación estandarizada para una arquitectura que puede manejar componentes y servicios de manera dinámica, originalmente diseñado para sistemas limitadas en recursos (i.e. embedded).

OSI de sus siglas en inglés Open Systems Interconnection. Es un modelo de referencia de interconexión de sistemas abiertos creado por la *→ISO*. Describe una red basada en varias capas de abstracción desde el nivel físico al nivel de una aplicación.

PCA análisis de componentes principales, por sus siglas en inglés Principal Component Analysis.

PID de sus siglas en inglés Process Identifier. El identificador único de un proceso en un sistema, en este caso una red local con varios nodos.

polimorfismo cualidad de lo que tiene o puede tener distintas formas. En la programación orientado a objetos se refiere al hecho de que un objeto puede tener varias clases (herencia) o implementar varias interfaces y por lo tanto ser utilizado por cada clase o interfaz.

POO programación orientada a objetos

protección medio por el cual se contiene un comportamiento peligroso de un sistema físico o por el cual se evitan las consecuencias del comportamiento (Isermann y Ballé, 1997).

respiración endógena respiración de oxígeno bajo ausencia de sustratos en el ambiente.

respiración metabólica respiración de oxígeno mediante la degradación metabólica de un sustrato en el ambiente.

ROM de sus siglas en inglés Reliable Ordered Multicast. Multicast con un orden total de los mensajes transferidos y una transmisión confiable (e.g. fallas de omisión de mensajes, que se pueden detectar).

RT tiempo real, de sus siglas en inglés Real Time. Requiere una definición sobre el comportamiento del tiempo de la transmisión de paquetes de datos mediante una red. En este trabajo se toman en cuenta las definiciones de Jasperneite y Watson (2003) y de Neumann (2006).

síntoma cambio de una cantidad observable del comportamiento nominal (Isermann y Ballé, 1997).

software conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.

supervisión monitoreo de un sistema físico tomando acciones adecuadas para mantener la operación en presencia de fallas (Isermann y Ballé, 1997).

TCP de sus siglas en inglés Transmission Control Protocol. Es un protocolo al nivel de transporte del modelo *(→ OSI)*, el cual establece una conexión con control de flujo y mecanismo de confirmación para el intercambio de paquetes de datos.

transacción interacción con una estructura de datos (→ *espacio de tuplas*) que consiste en una serie de operaciones básicas que se deben aplicar como una operación indivisible delante de

- fallas;
- procesos que estén accediendo simultáneamente a los datos.

TS de sus siglas en inglés Tuple Space. → *espacio de tuplas*

tupla un conjunto que contiene un número arbitrario de campos con un tipo primitivo e.g. {1; B1; 1, 194} es una tupla con la estructura $\{int, string, float\}$.

UML lenguaje unificado de modelado; por sus siglas en inglés Unified Modelling Language. Es un lenguaje estandardizado para el modelado de sistemas de software orientado a objetos, el cual permite visualizar, especificar, construir y documentar el sistema.

unicast envio de un mensaje de un nodo a otro por medio de una red.

UDP de sus siglas en inglés User Datagram Protocol. Es un protocolo al nivel de transporte del modelo → *OSI* el cual consiste en el intercambio de paquetes de datos sin control de flujo o mecanismo de confirmación.

Apéndice G

Publicaciones

Revista internacional y arbitrada

D. Wimberger and C. Verde. Fault diagnosticability for an aerobic batch wastewater treatment process. Control Engineering Practice, 2008. 10.1016/j.conengprac.2008.03.002 (in press).

D. Wimberger and C. Verde. Diagnosis of abnormal conditions of an aerobic sbr process. Mathematical and Computer Modelling of Dynamical Systems, 14(1):53–66, February 2008.

Conferencia internacional

Arbitrada

D. Wimberger and C. Verde. Online monitoring of an aerobic SBR process based on dissolved oxygen measurement. In Proceedings of the 16th IFAC World Congress, Prague, Czech Republic, July 3-8 2005. Elsevier. ISBN: 0-08-045108-X.

Sesión invitada

D. Wimberger and C. Verde. Detection of abnormal conditions of an aerobic sbr process using feature extraction. In I. Troch and F. Breitenecker, editors, Proceedings of the 5th MATHMOD Conference, February 2006.