Universidad de Deusto (tercer ciclo)

Programa: Ciencia de la Computación e Inteligencia Artifical

METODOLOGÍA CON REUTILIZACIÓN SISTEMÁTICA PARA EL DESARROLLO DEL SOFTWARE DE CONTROL DE FMSs

Tesis doctoral presentada por Dña. Mª Isabel Sarachaga González

Dirigida por el Dr. Evaristo Kahoraho Bukubiye

El Director

El Doctorando

Bilbao, Septiembre de 1.998

A mi familia

Mi más sincero agradecimiento al Director de esta tesis Dr. Evaristo Kahoraho, y al Departamento de Ingeniería Mecánica de la Escuela Técnica Superior de Ingeniros de Bilbao, en especial a José Ignacio Llorente y a mis compañeros Arantza Burgos y José Antonio Sánchez.

Índice

Pág.

Índice de figuras

Pág.

Índice de tablas

Pág.

Resumen

En el presente trabajo se propone una metodología con reutilización sistemática para la construcción de sistemas de control de FMSs. Concebida bajo el enfoque de la Megaprogramación, sus principales características son: reutilización, visión de dominio, arquitectura genérica para la familia de productos, guías para el proceso de desarrollo y librerías de elementos reutilizables.

Esta nueva metodología establece un ciclo de vida iterativo e incremental, que comprende dos procesos independientes pero cooperantes (ingeniería de dominio e ingeniería de aplicación) para construir el software de control de FMSs. La ingeniería de dominio se centra en la construcción de la familia de productos, mientras que la ingeniería de aplicación persigue el desarrollo de sistemas de control particulares.

La bondad de la metodología propuesta se ha evaluado en un proyecto piloto desarrollado en el Departamento de Ingeniería Mecánica de la Escuela Técnica Superior de Ingenieros de la UPV/EHU (Universidad del País Vasco / Euskal Herriko Unibertsitatea), en el bienio 1.996-97. El análisis de la interacción entre la ingeniería de dominio y la ingeniería de aplicación ha proporcionado unos resultados satisfactorios.

Abstract

In this work a methodology with a systematic reuse for the development of FMS control systems is proposed. This methodology is considered under the Megaprogramming approach, and its main features are: domain specific reuse, a generic architecture for the product family, a set of well defined procedures for using the domain products to build application systems, and a repository of reusable components.

The new methodology has an iterative and incremental life cycle, which establishes two independent although collaborative processes (domain engineering and application engineering), to develop the FMS control software. Domain engineering focuses on creating the product family. whereas application engineering has a particular product as a result.

The goodness of the methodology proposed has been evaluated with a pilot project, which has been developed in the Department of Mechanical Engineering at the College of Engineering (University of the Basque Country), during 1.996-1.997. The interaction between domain engineering and application engineering has been evaluated and satisfactory results have been obtained.

Laburpena

FMS kontrol sistemen garaketarako berrerabilera sistematikoaren metodologia proposatzen da lan honetan. Megaprogramazioaren filosofian oinarriturik, hauek dira bere ezaugarri aipagarrienak: berrerabilera, esparrura zuzendua, produktuen familiarako arkitektura generikoa, garaketa prozesurako gidak eta elementu berrerabiligarrien liburutegien erabilera.

Metodologia berri honek elkarreraginezko eta inkremental bizi-ziklo bat finkatzen du. Bizi-ziklo hau FMS kontrol-software garaketarako bi prozesu independiente eta lagungarriz osaturik dago, hain zuzen ere, esparru-ingeniaritza eta aplikazio-ingeniaritza. Esparru-ingeniaritza produktuen familiaren eraiketan zentratzen da; aldiz, aplikazio-ingeniaritzak kontrol-sistema ezagun baten garaketa du helburu.

Proposatutako metodologia honen emaitzak ebaluatuak izan dira proiektu piloto bat erabiliz. Proiektu hau UPV-EHU-ko Bilboko Ingenierien Goi-Eskolan dagoen Mekanika Ingeniaritza Sailan garatu da, 1.996 eta 1.997 urteetan zehar. Esparru-ingeniaritza eta aplikazio-ingeniaritzaren arteko elkarrakzioaren analisiaren emaitzak oso baliogarriak izan dira.

Introducción

Una **metodología para el desarrollo del software de control de FMSs, que incorpore la reutilización de forma sistemática** desde las primeras etapas de su proceso de construcción, es la solución que se propone en el presente trabajo **para solventar las deficiencias de las instalaciones actuales:**

- *Implementación de islas de automatización que exhiben un bajo nivel de integración.*
- *Funcionalidad limitada por la imposibilidad de integrar componentes de distintos fabricantes como resultado de interfaces propietarias.*
- *Mínima reutilización de componentes software repercutiendo en mayores costes y tiempos de puesta en marcha.*
- *Diferencias arbitrarias en las técnicas de implementación.*
- *Ausencia de herramientas de desarrollo comunes.*

Obviamente, precisa gran cuidado la construcción de este tipo de software, responsable de la planificación, el control y el seguimiento de todas las actividades desempeñadas en el sistema, así como de la integración de éste con las restantes funciones productivas de la empresa *[Lynggaard, 96]*. Hay que tener presente que este software afecta en gran medida a los principales atributos de un sistema de fabricación: coste, tiempo, calidad y flexibilidad.

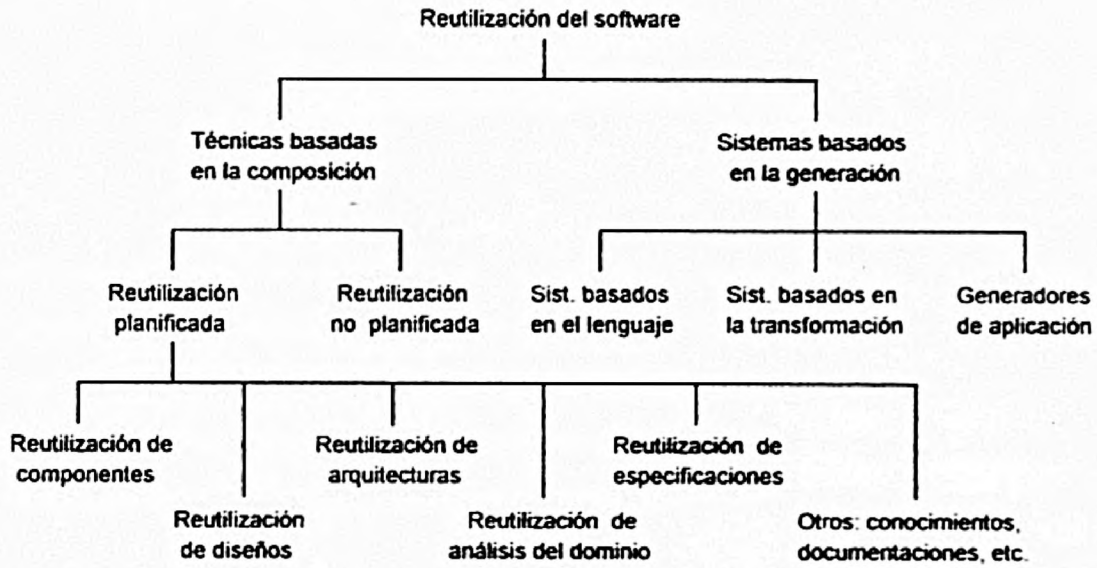
El análisis de los procesos de un proyecto de construcción del software de control de un FMS es generalmente mucho más complejo que las aplicaciones de gestión clásicas. Esto es debido a las características especiales de las tecnologías de la información cuando se integran en el entorno industrial: su labor no consiste en transformar información en otra información, sino **en transformar información en acciones de dispositivos de un modo totalmente controlado** *[Waldner, 92]*.

Existen numerosas metodologías que abordan el desarrollo de estos sistemas, pero ninguna de ellas incorpora la reutilización en el propio proceso de construcción del software. Cuando presentan la reutilización de software como uno de sus principales

beneficios, normalmente se refieren a la reutilización de código ejecutable (fragmentos de código, subrutinas o módulos).

Con tal definición se limitan los beneficios potenciales, reduciéndolos a un pequeño incremento de la productividad del programador. Sin embargo, en el contexto más amplio, el concepto de reutilización afecta a conceptos, herramientas, análisis, diseño, especificaciones, documentación y conocimiento del dominio, además de componentes.

Desde este punto de vista, se puede realizar una taxonomía de la reutilización del software. La que se propone combina las clasificaciones propuestas por Biggerstaff et al. *[Biggerstaff, 89]* y Krueger *[Krueger, 92]*.



Taxonomía de la reutilización del software.

En esta taxonomía se distinguen **dos categorías** generales claramente diferenciadas por la forma en que realizan la reutilización: las técnicas basadas en la composición construyen el software basándose en componentes, mientras que los sistemas basados en la generación reutilizan software en el proceso de generación de otros sistemas.

En las **técnicas basadas en la composición** se distingue: *la reutilización no planificada*, que es la empleada habitualmente cuando se usan fragmentos de código, y *la reutilización planificada*, que engloba varios casos según el tipo de elemento que se reutilice.

Los **sistemas basados en la generación** abarcan: *los sistemas basados en el lenguaje*, cuyos compiladores reutilizan patrones en lenguaje ensamblador, *los sistemas basados en la transformación*, que reutilizan la secuencia de transformaciones realizadas a la especificación para mejorar la eficiencia en ejecución, y *los generadores de aplicación*, que asisten en la construcción de software dentro de dominios particulares.

La reutilización planificada (a todos los niveles) y los generadores de aplicación son los aspectos que debe incorporar la nueva metodología para obtener una reutilización sistemática, y con ello, garantizar totalmente los beneficios derivados de la reutilización.

La nueva metodología con reutilización sistemática propuesta, el estado del arte y los resultados obtenidos se presentan en cuatro capítulos de la forma siguiente:

- **Primer capítulo.** Se analizan **las técnicas de Ingeniería de software empleadas hasta la fecha** en el desarrollo de los sistemas de control de FMSs. Se describen las principales arquitecturas genéricas, métodos estructurados y métodos orientados a objetos, aportándose un análisis de los mismos desde la perspectiva de la reutilización, y se presentan las aplicaciones más relevantes.
- **Segundo capítulo.** Se analiza **la Megaprogramación como nueva tendencia en el desarrollo de software**, y se presenta una **visión general de la metodología con reutilización sistemática para el desarrollo del software de control de FMSs aportada.** Esta metodología se ha desarrollado bajo el enfoque de la Megaprogramación.
- **Tercer capítulo.** Se describe detalladamente **la nueva metodología, distinguiendo los dos procesos, ingeniería de dominio e ingeniería de aplicación,** involucrados en la construcción del software.
- **Cuarto capítulo.** Se presenta **el proyecto piloto** que permite evaluar la bondad de la metodología propuesta en aplicaciones industriales.

En las **conclusiones,** se destacan los resultados obtenidos y la aplicabilidad industrial de esta metodología. Después se presentan **las principales aportaciones** y se definen **futuras líneas de investigación.**

Por otro lado, en el documento se emplean numerosos vocablos en inglés, que no han sido traducidos para evitar la alteración o pérdida de su significado. Por ello, en los apéndices A y B se han desarrollado respectivamente un glosario de términos específicos y un glosario de acrónimos para el dominio del software de control de FMSs.

Capítulo 1

Técnicas actuales de Ingeniería de Software para el desarrollo del software de control de FMSs

1.1 Introducción.

Antes de proponer nuevas soluciones al desarrollo del software de control de FMSs, hay que **analizar las soluciones adoptadas hasta la fecha desde la perspectiva de la reutilización del software;** objetivo éste del presente capítulo.

Los esfuerzos de grandes compañías, grupos de investigación y comités de estandarización se centran en la obtención de **modelos genéricos de fabricación,** los cuales establecen los niveles funcionales de control, y las pautas para llevar a cabo la gestión de desarrollo y operativa del software. Esto significa que cuando el desarrollo se apoya en un modelo de referencia no se construye partiendo de cero, sino que se realiza **una instanciación a través de la especialización del modelo.**

Pero cualquiera de estos modelos de referencia no es suficiente a la hora de llevar a cabo la implementación, requiriendo el soporte de las Tecnologías de la Información e Ingeniería del Software.

El empleo inicial de **métodos estructurados** para el desarrollo de estos sistemas, revierte en componentes modulares específicos de la aplicación concreta, con **escaso grado de reutilización** a costa de gran esfuerzo.

Posteriormente, se aplican **métodos orientados a objetos,** los cuales establecen una estrecha correspondencia entre las entidades reales, las diseñadas y las implementadas, favoreciendo así la incorporación de la reutilización. Los resultados obtenidos incrementan la **reutilización en mayor o menor grado,** pero la reutilización no es parte imbricada en el proceso de construcción.

El análisis de estas soluciones de ingeniería precisa el establecimiento de unos criterios que permitan realizar una comparativa. A la hora de **estudiar el impacto que las metodologías de desarrollo tienen en la reutilización** del software, se analizan tres **aspectos** claves:

- **Construcciones básicas.** Las metodologías disponen de unos conceptos o construcciones básicas a partir de los cuales se desarrollan las aplicaciones (clase, mensajes, módulos, etc.).

- **Proceso.** Cada metodología prescribe una serie de pasos a realizar que pueden abarcar total o parcialmente el ciclo de vida de un desarrollo de software.
- **Documentación.** Este aspecto hace referencia a la notación gráfica que propone cada metodología, y no tanto a la documentación textual soportadas por defecto en todas ellas.

Indudablemente, *una metodología que soporte la reutilización del software debe disponer de unas construcciones básicas que faciliten la reutilización. Su proceso debe integrar este concepto y proporcionar guías explícitas para el desarrollo para con reutilización, así como para la gestión de los componentes reutilizables. Su notación gráfica debe proporcionar una documentación clara que permita entender los componentes reutilizables.*

El capítulo se ha dividido en cinco secciones a lo largo de las cuales se analizan las técnicas actuales de Ingeniería de software para el desarrollo del software de control de FMSs.

En la **segunda sección** se describen las **arquitecturas genéricas** aplicables al desarrollo del software de control de FMSs, se presentan las más representativas de los tipos identificados, y se aporta el análisis que éstas tienen en la construcción de un sistema de control desde la perspectiva de la reutilización.

En la **tercera sección** se describen los **métodos estructurados** más utilizados en la actualidad, se aporta el análisis de los mismos respecto a la reutilización, y se presentan algunos ejemplos de aplicación de estos métodos al desarrollo de sistemas de control.

En la **cuarta sección** se presentan los **métodos orientados a objetos** más relevantes, se aporta el análisis de estos métodos desde la perspectiva de la reutilización, y se describen algunas aplicaciones de esta tecnología al desarrollo de software de control para sistemas de fabricación.

En la **quinta sección** se exponen las **conclusiones,** resaltando la necesidad de una metodología para el desarrollo de software de control de FMSs que integre la reutilización en el propio proceso de construcción.

1.2 Arquitecturas genéricas para el software de control de FMSs.

1.2.1 Clasificación general.

El consorcio AMICE *[AMICE, 93]* define arquitectura en los siguientes términos:

> *"Una arquitectura es la estructura y diseño de algo; una colección de elementos que permiten estructurar y diseñar algo de forma consistente".*

Bajo este prisma, una arquitectura para el software de control de FMSs constituye una herramienta de soporte para que los desarrolladores de sistemas de fabricación automatizados mejoren la estructura de sus diseños, y por ende, la implementación del sistema.

Entre otros autores, Williams *[Williams, 97]* indica que existen **dos tipos de arquitecturas** en relación con la integración de entidades de fabricación o empresas:

- **Arquitecturas de referencia de empresa.** Se emplean para organizar el desarrollo y la implementación de un sistema global de integración de empresa.
- **Arquitecturas para el diseño de sistemas físicos.** Son arquitecturas asociadas a un tipo de sistema específico, de las cuales se destacan las *arquitecturas CIM* y las *arquitecturas funcionales de control* por su interés para el presente trabajo.

Ambos tipos son de gran **ayuda a la hora de construir el sistema de control de un FMS** *[Joshi, 94]*, sobre todo en lo concerniente a los siguientes aspectos:

- **Especificación.** Una arquitectura proporciona una estructura que identifica todas las actividades y componentes de un FMS, así como las relaciones entre ellos.
- **Ciclo de vida.** Si la arquitectura describe explícitamente las etapas de construcción, también asistirá al desarrollo del sistema de control de un FMS, indicando las tareas a realizar y el momento para realizarlas.

- **Análisis.** Dado que una arquitectura representa la funcionalidad del sistema, se puede emplear este modelo para hacer simulaciones del sistema de control con varios niveles de detalle.

- **Operación.** Los modelos funcionales de una arquitectura contienen la lógica de control; por tanto, se podrán usar dichos modelos para el control del FMS.

Por otra parte, las **arquitecturas funcionales de control** se ciñen más al problema de los sistemas de control de FMSs. Su propósito es conseguir que todos los componentes del entorno de fabricación trabajen de forma integrada para obtener productos de calidad a un coste razonable. Dichas arquitecturas *identifican los componentes, especifican reglas de integración, y presentan interfaces estándares para construir componentes que puedan interoperar.*

Tomando como referencia la definición del consorcio AMICE, Maglica *[Maglica, 97]* apunta que una arquitectura de este tipo debe tener las siguientes características:

- **División funcional.** La arquitectura debe englobar un conjunto de entidades funcionales bien identificadas, con objeto de obtener una implementación modular. La correcta definición de los interfaces entre las mismas revertirá en la construcción de módulos software intercambiables. Además, la obtención de un sistema estable precisa que estas entidades sean autónomas y cooperantes, en la línea de la fabricación holónica *[Sugimura, 95], [Tönshoff, 95].*

- **Separación de los aspectos genéricos y específicos.** La arquitectura debe identificar aquellas partes genéricas aplicables a todo el colectivo de los sistemas de fabricación y aquellas otras específicas de sistemas particulares. La no consecución de esta característica conlleva una reducción de la reutilización de software y de las tareas de ingeniería en general.

- **Adaptación al entorno de trabajo.** La arquitectura debe usar una estructura y terminología que tenga sentido en el dominio de las plantas de fabricación. El caso omiso de este punto, probablemente degradará la construcción del sistema

debido a los problemas de integración del conocimiento del personal involucrado (operarios, ingenieros de producción, etc.).

- **Simplicidad.** Una arquitectura simple es más sencilla de comprender y aplicar. Esta característica es parcialmente contradictoria a otras comentadas previamente, y por lo tanto, debe establecerse un compromiso entre ellas.

En los apartados siguientes se presentan las arquitecturas más relevantes de los tipos identificados, dada su influencia en la construcción de los sistemas de control de FMS. Finalmente, se analiza el impacto que las arquitecturas genéricas tienen en la construcción del software de control desde el punto de vista de la reutilización.

1.2.2 Principales arquitecturas actuales.

1.2.2.1 Arquitecturas de referencia de empresa.

Una arquitectura de referencia de empresa *modela el ciclo completo de un proyecto de integración al nivel de empresa*, desde su concepción inicial pasando por su definición, diseño funcional, diseño detallado, implementación y puesta en explotación, hasta su obsolescencia.

En el ámbito de la Ingeniería de empresa[1] se han desarrollado recientemente muchas arquitecturas y metodologías; algunas tienen sus orígenes en los sistemas CIM. En la tabla 1.1 se presentan las arquitecturas de referencia de empresa más destacables.

---

[1] La Ingeniería de empresa comprende el modelado, análisis, diseño e implementación de sistemas integrados de empresa *[Toh, 97]*.

ARIS (ARchitecture for Information Systems) de la empresa IDS Scheer
ARIS se centra en el modelado de la empresa desde la perspectiva operativa y de las tecnologías de la información, con objeto de construir el sistema de información de la empresa *[Kosanke, 97]*.
CIM-OSA (CIM Open System Architecture) del programa ESPRIT
Esta arquitectura abierta aporta: una definición general del alcance del CIM, una guía de implantación, una descripción de los sistemas y subsistemas componentes, y una estructura modular que acata los estándares internacionales *[AMICE, 93]*, *[Goenaga, 91]*, *[Kusiak, 92]*. El modelo de referencia CIMOSA ha sido validado en diferentes sectores por sucesivos proyectos ESPRIT: VOICE en el sector del automóvil, TRAUB en el sector de la máquina herramienta, y VOICE II en el sector de fundición de aluminio *[CIMOSA, 96]*, *[VOICE II, 95]*.
GRAI-GIM (GRAI Integrated Methodology) del laboratotio GRAI
Inicialmente fue desarrollada para modelar la estructura de decisiones de una empresa manufacturera en lo referente a la planificación estratégica, táctica y operativa. Posteriormente fue ampliada para soportar el diseño de sistemas CIM, convirtiéndose en una metodología integrada para el modelado de procesos de negocio, que establece un enfoque estructurado claro para el análisis y desarrollo de este tipo de sistemas *[Chen, 96]*, *[Joshi, 94]*.
IEM (Integrated Enterprise Modelling) del IPK Fraunhofer Institute
IEM soporta la creación de modelos de empresa para la realización de reingeniería de procesos, y por tanto, también permite modelar la dinámica de los procesos con el fin de evaluar alternativas operativas *[Kosanke, 97]*.

Tabla 1.1: Arquitecturas de referencia de empresa.

PERA (Purdue Enterprise Reference Architecture) del laboratorio Purdue Applied Control
Este modelo incluye el ciclo de vida más completo para el desarrollo de sistemas CIM. El aspecto más destacable es que involucra explícitamente la organización y el personal en el modelo, investigando este factor tan crítico en la integración del sistema total */Chen, 96/, /Joshi, 94/.*
La Rueda de CASA/SME (Computer and Automated Systems Association/ Society of Manufacturing Engineers)
La Nueva Rueda de la Empresa Manufacturera es un índice gráfico del conocimiento y enfoque organizativo preciso en el ámbito de fabricación actual. Describe seis elementos fundamentales para obtener una fabricación competitiva: los clientes, el personal y equipo de la organización, los sistemas y el conocimiento compartido, los procesos, los recursos y responsabilidades, y la infraestructura de fabricación */CASA SME, 93/.*
GERAM (Generalised Enterprise Reference Architecture and Methodology) de IFAC/IFIP (International Federation of Automatic Control/International Federation for Information Processing) Task Force
GERAM define aquellos métodos, modelos y herramientas necesarios para desarrollar, diseñar, construir y mantener una empresa integrada (de cualquier tipo) en un entorno cambiante */Bernus, 94/.* A diferencia de las anteriores, GERAM no es una propuesta más, sino que organiza todo el conocimiento existente en integración de empresas en lugar de redefinirlo. De este modo, las arquitecturas previas mantienen su identidad, pero pueden contrastar los beneficios que solapan y complementan con otras arquitecturas a través de GERAM.

Tabla 1.1 (continuación).

1.2.2.2 Arquitecturas para el diseño de sistemas físicos. Arquitecturas CIM.

La mayoría de las arquitecturas CIM *realizan la formalización sobre la base de un modelo determinista y jerárquico de planificación y control.* En la tabla 1.2 se describen brevemente las más representativas.

FAM (Factory Automation Model) de ISO (International Standards Organization)
FAM establece los conceptos de un modelo genérico para una planta de producción en una estructura jerárquica de seis niveles (Empresa, Fábrica/Planta, Sección/Área, Célula, Estación, Equipo). En cada nivel se descomponen los objetivos del nivel superior en tareas más sencillas, se asignan tareas y recursos a sus subordinados, y se evalúa la información procedente del nivel inferior *[Bauer, 94], [Deregibus, 91], [Kusiak, 92].*
AMRF (Advanced Manufacturing Research Facility) de NBS (National Bureau of Standards)
Desarrollada exprofeso con hardware y software de diversos fabricantes, propone una estructura jerárquica de cinco niveles (Fábrica, Área, Célula, Estación y Equipo). Las tareas de los niveles superiores se descomponen en secuencias de sub-tareas de forma sucesiva hasta llegar al nivel más bajo de la jerarquía donde se ejecutan acciones simples *[Bauer, 94], [Deregibus, 91].*
Modelo del programa ICAM (International Computer Aerospace Manufacturing) de la U.S. Air Force
Considerado como uno de los proyectos más ambiciosos, persigue el desarrollo de métodos estructurados para aplicar las tecnologías basadas en el ordenador a la fabricación, con objeto de mejorar la productividad. El modelo resultante establece una estructura funcional de cinco niveles *[Deregibus, 91].*

Tabla 1.2: Arquitecturas CIM.

COPICS (Communication-Oriented Production Information and Control System) de IBM
Este modelo recoge las principales actividades en cuanto a control y planificación de la fabricación, haciendo especial énfasis en las comunicaciones, la gestión de bases de datos y las presentaciones */Rembold, 93/.*
Modelo de Digital Equipment Corporation
Este enfoque divide el sistema de control en módulos funcionales con sus correspondientes flujos de datos, constituyendo esta información la base para establecer el layout de la planta. Como resultado se obtienen los modelos funcional y físico del sistema de fabricación y de sus subsistemas */Rembold, 93/.*
Modelo de Siemens
Este modelo presenta las principales funciones técnicas y organizativas para la fabricación, integrando todas ellas a través del flujo de información. Se distingue un flujo de información vertical necesario para interconectar los distintos niveles del modelo jerárquico de la empresa, y un flujo de información horizontal para controlar y sincronizar las actividades paralelas de cada nivel */Rembold, 93/.* Además, incorpora los conceptos de la organización de la empresa asistida por ordenador (CAO – Computer Aided Organization), que comprende las áreas de contabilidad, personal y contabilidad industrial, y de la industria asistida por ordenador (CAI – Computer Aided Industrie), que engloba CIM y CAO.

Tabla 1.2 (continuación).

1.2.2.3 Arquitecturas para el diseño de sistemas físicos. Arquitecturas funcionales de control.

Las arquitecturas de este tipo *asignan las responsabilidades de la toma de decisiones a componentes específicos del sistema de control, y determinan las relaciones entre dichos componentes.* A continuación se presentan algunos ejemplos.

Ninguna de las arquitecturas funcionales que se describen constituye un estándar; ni siquiera alguna de ellas se perfila como tal. Pero indudablemente, pese a sus limitaciones, *son un buen punto de partida para acometer un proyecto de automatización para un FMS.*

1. Arquitectura propuesta por Bauer

El **proyecto COSIMA** (COntrol Systems for Integrated MAnufacturing) *[Bauer, 94]* remarca la necesidad de una arquitectura flexible y genérica que identifique y delimite las funciones correspondientes a *un sistema de control de las actividades de producción (PAC - Production Activity Control) de cada célula; necesidad que se extrapola a un controlador de nivel superior (FC - Factory Coordination) que coordine las actividades de varias células.*

Bauer et al. presentan un **sistema PAC** que proporciona las funciones necesarias para controlar el flujo de producción mediante la interacción de cinco componentes:

- **Scheduler.** Desarrolla planes según las guías contenidas en el plan de nivel superior.
- **Dispatcher.** Controla el flujo de trabajo dentro de la planta en tiempo real.
- **Monitor.** Observa el estado de la planta y transmite cualquier información relevante al scheduler, dispatcher y nivel superior.
- **Mover.** Organiza el movimiento de materiales entre estaciones de trabajo.
- **Producer.** Controla la secuencia de operaciones en cada una de las estaciones.

La **arquitectura del FC** comprende dos tareas:

- **Diseño del entorno de producción.** Reorganiza el layout basándose en el producto para mantener su eficiencia. Para ello, usa una selección de técnicas concernientes a la planificación del proceso, el mantenimiento de un layout basado en el producto y el análisis del sistema de fabricación.

- **Control.** Coordina el flujo de productos para garantizar que se cumplen las fechas de producción con altos niveles de calidad y bajos costes. Esto se obtiene mediante los correspondientes módulos de Scheduler, Dispatcher y Monitor, encargados de comunicar las guías apropiadas a los sistemas PAC individuales.

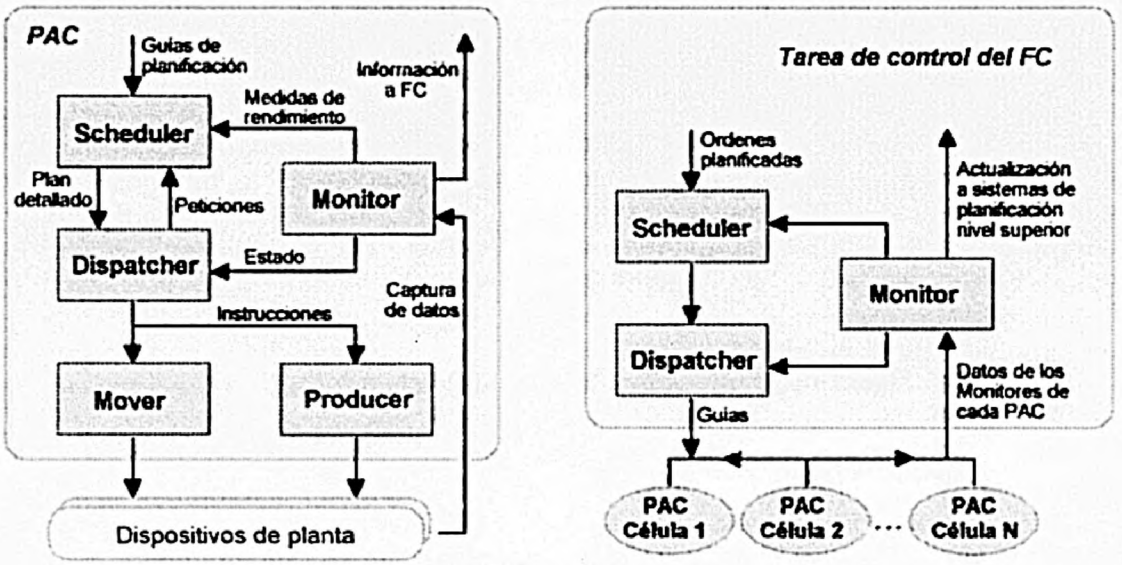


Figura 1.1: Arquitectura propuesta por Bauer.

2. Modificación propuesta por Maglica a la arquitectura de Bauer

Maglica *[Maglica, 97]* analiza la arquitectura de célula propuesta en el proyecto COSIMA y concluye que *adolece de dos deficiencias: el manejo de la información está muy centralizado debido a la definición del módulo Monitor, y la proyección del PAC en el sistema físico de fabricación es débil por causa de la definición de los módulos Mover y Producer.*

Para solventar estos problemas, *Maglica propone la división de los módulos Mover y Producer en varias entidades de estos tipos* (una por workstation de planta), *y distribuir el tratamiento de la información realizado por el Monitor con estas nuevas entidades.*

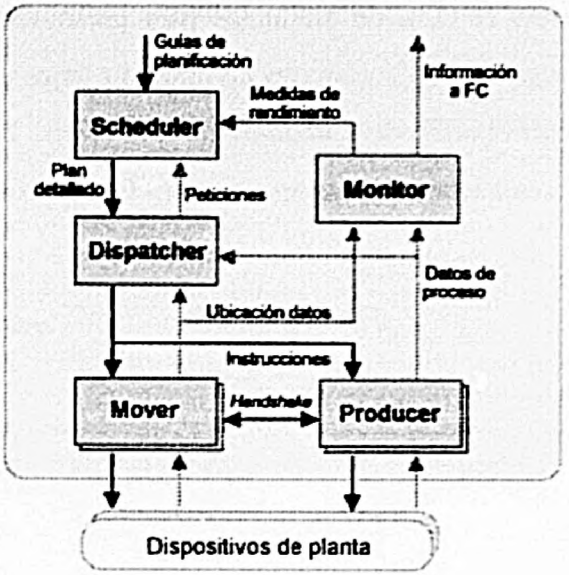


Figura 1.2: Arquitectura PAC modificada por Maglica.

3. Arquitectura propuesta por Bakker

La **arquitectura del controlador de célula distribuido** propuesta por Bakker */Bakker, 91/* presenta un *sistema de control distribuido lógica y físicamente en una serie de componentes cooperantes:*

- Cada estación dispone de un componente denominado **Gestor de estación,** que determina la secuencia de operaciones en la máquina y garantiza la disponibilidad de los recursos necesarios para procesar las operaciones.
- Los **Módulos de función** proporcionan los recursos solicitados por los Gestores de estación. Estos Módulos de función son responsables de la gestión de las herramientas, de los palets y de los programas, asi como del transporte y de la carga de nuevos productos al sistema.

Una característica reseñable de esta arquitectura es la ausencia de plan, sustituido éste por una lista de operaciones asociada a cada producto. Cuando una máquina concluye una operación en un producto, su Gestor de estación comprueba si el producto requiere más operaciones. En caso afirmativo, el Gestor de estación negocia con los restantes la adjudicación de la operación, que se asignará a aquella máquina cuya cola de espera sea más corta.

La principal limitación que se observa en esta arquitectura es la rigidez de la estrategia de trabajo (única). Presupone que todas las piezas correspondientes a un periodo de planificación tiene la misma fecha de entrega. Sin embargo, si fuera posible prescindir de las fechas de entrega, esta estrategia sería muy eficiente y sencilla de aplicar e implementar.

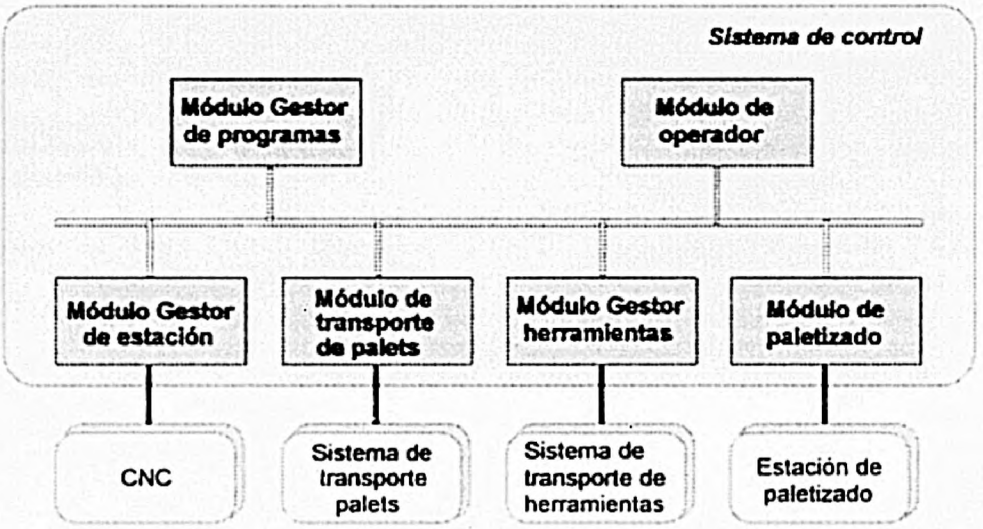


Figura 1.3: Arquitectura propuesta por Bakker.

4. Arquitectura propuesta por Meyer

Una arquitectura de especial interés es la planteada por Meyer *[Meyer 91]*. Ésta introduce el concepto de **"Knowledge-based CIM"**, resaltando así la necesidad de integrar el conocimiento, frente a la simple interconexión de equipos o compartición de información en bases de datos.

Este controlador CIM, *dotado de un cierto grado de inteligencia, configura un sistema de control dividido en una red de información y una red de decisiones sobre la planta física.* Los módulos identificados realizan las siguientes tareas:

- **Interpretación.** Enlace entre la observación del mundo real y el modelo que describe dicho mundo.
- **Diagnóstico.** Explicación de la desviación entre la situación actual y la situación deseada para el sistema.

- **Planificación de acciones.** Configuración óptima de acciones para lograr o mantener una meta, lo cual presupone la existencia de una serie de acciones elementales.

- **Toma de decisiones** (habitualmente en tiempo real). Periodo de tiempo definido por los tiempos de reacción del control, los intervalos de decisión y los periodos de planificación.

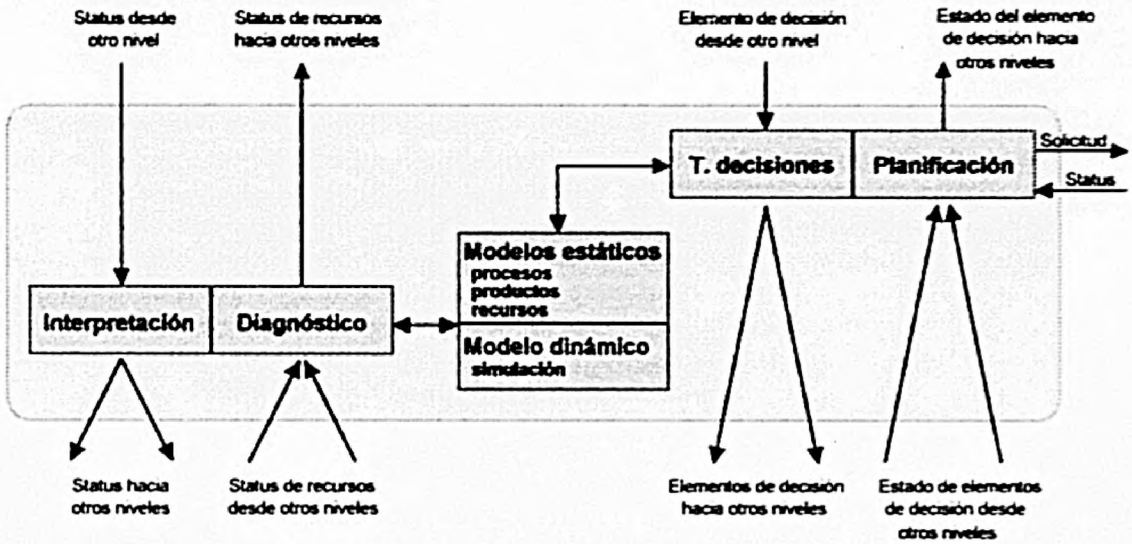


Figura 1.4: Arquitectura propuesta por Meyer.

5. Arquitectura propuesta por la Universidad de Twente

El Laboratorio de Ingeniería de Diseño y Producción de la Universidad de Twente (Países Bajos) ha desarrollado un concepto genérico para el control de planta: **SFC (Shop Floor Control)** *[FACT, 97]*. Este concepto *se emplea para determinar tanto la organización de las actividades de fabricación, como el modo de desarrollar los sistemas SFC que coordinan la ejecución de las tareas.*

La **arquitectura funcional** que propone consta de *seis módulos distribuidos en dos niveles jerárquicos:*

- **Módulos Scheduling, Dispatching, Monitoring y Diagnostics en el nivel superior.** Estos módulos llevan a cabo las funciones de planificación, control y

monitorización necesarias para ejecutar en plazos y de modo eficiente las tareas contenidas en el plan de producción.

- **Módulos Control de estación y Control de estación auxiliar en el nivel inferior.** Módulos a cargo de controlar la ejecución de las tareas de producción propiamente dichas y de las tareas auxiliares (preparación, almacenamiento y localización de herramientas, fijaciones, elementos de medida, tareas de inspección, etc.).

El concepto de "Estación Virtual", implementado en los módulos Control de estación y Control de estación auxiliar, *permite la construcción de sistemas SFC independientes del nivel de automatización del equipamiento de fabricación.*

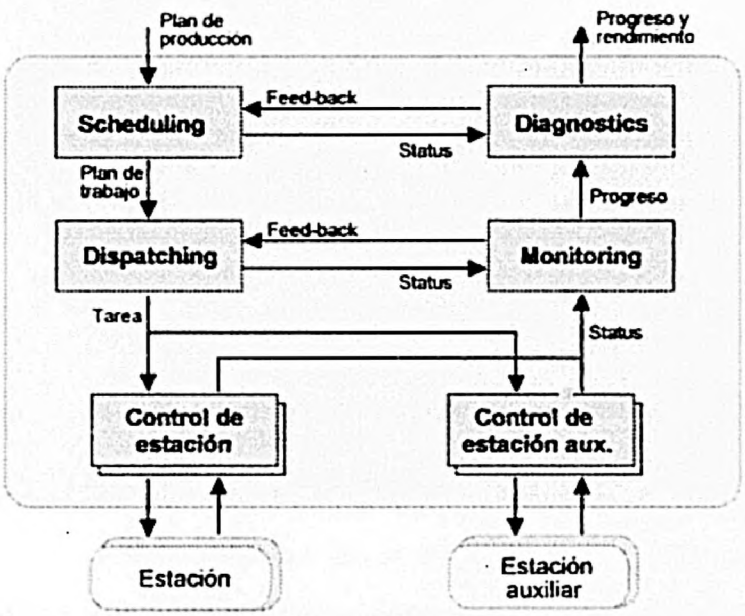


Figura 1.5: Arquitectura propuesta por la Universidad de Twente.

6. Arquitectura propuesta por Adiga

Adiga presenta una **arquitectura orientada a objetos para aplicaciones CIM** *[Adiga, 93]*, que *divide todo sistema de fabricación en dos partes: el sistema de decisiones y el sistema físico.* También indica que cualquier sistema de decisiones típico se identifica como una combinación de los siguientes módulos o subsistemas:

- **Modelo de estado.** Modelo computerizado de la planta de fabricación que refleja el estado de la misma, aportando el contexto para la toma de decisiones.
- **Control.** Agrupa la lógica de control empleada en la resolución de los problemas de la aplicación. Para llevar a cabo sus funciones de monitorización, planificación, control, etc. trabaja sobre el modelo de estado.
- **Interface entre el modelo de estado y la planta.** Permite definir el modelo de estado y adquirir la información del sistema físico.
- **Interface entre el módulo de control y la planta.** Recibe las peticiones de usuario o del mundo real en general, y envía las decisiones o recomendaciones.

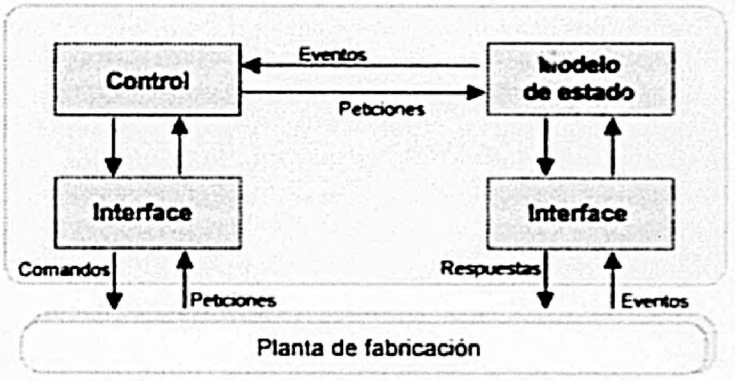


Figura 1.6: Arquitectura propuesta por Adiga.

Esta arquitectura ha sido *concebida para proporcionar un alto grado de modularidad y reutilización.* La interacción y extensión de sus módulos se garantiza mediante un *bus de mensajería software*, al cual se conectan los módulos como si de tarjetas se tratara.

7. Arquitectura propuesta por Robotiker

La **arquitectura funcional propuesta por Robotiker** *[Robotiker, 92]* identifica los siguientes *módulos software: Monitor, Mantenimiento, Scheduler dinámico, Control básico Dispatcher, Gestores de recursos, Servidores de dispositivos, Bus de mensajes, y Sistemas de comunicaciones industriales.*

En el presente trabajo se propone una versión modificada de esta arquitectura que será utilizada como arquitectura genérica del dominio en la metodología desarrollada.

1.2.3 Análisis del impacto de las arquitecturas genéricas en la reutilización.

Las **arquitecturas de referencia de empresa y las arquitecturas CIM** tienen un importante papel en la construcción del software de control, puesto que muchos de los datos necesarios para diseñar el nuevo sistema ya existen en dichas arquitecturas.

Toh et al. *[Toh, 97]* afirman que estas arquitecturas deben unirse a la creación de las especificaciones de los sistemas de información de planta. Sin contradecirle, destacar que el ámbito de estas arquitecturas es mucho mayor que el correspondiente a los sistemas de control, por lo que se deduce que *sólo se pueden reutilizar algunas partes de sus modelos, aunque sí su metodología de construcción.*

Sin embargo, las **arquitecturas funcionales de control** se pueden usar como diseños software genéricos aplicables en la construcción de diferentes sistemas, *fomentando así la reutilización de diseños, y por ende, la reutilización de componentes.*

Cuando se usa un modelo de este tipo, *se asume una arquitectura para el análisis y diseño del sistema.* Por lo tanto, se evita la descomposición estructurada sucesiva realizada por los métodos empleados tradicionalmente.

También *se eliminan los problemas clásicos de diseñar estos sistemas partiendo de cero:*

- *Acometer los mismos problemas en cada nueva implementación.*
- *Reconfigurar el sistema cuando surjan cambios en los procesos y/o productos.*
- *Integrar la información y funcionalidad a través de la empresa.*

Aunque no sea tan sencillo porque existen determinados procesos de adaptación, Bauer et al. *[Bauer, 94]* resumen en la siguiente frase el papel de estas arquitecturas:

> *"Cuando se usa una arquitectura, la tarea de implementar es el siguiente paso en la construcción del sistema real".*

Normalmente, la especificación de una arquitectura de control incluye:

- *Funcionalidad e interfaces de cada componente de la arquitectura.*
- *Organización de los componentes en el sistema.*
- *Mecanismos de comunicación entre los componentes.*
- *Modos de interacción entre los componentes.*
- *Especificación de la estructura de los planes del sistema de control.*
- *Especificación de las tareas desempeñadas por el sistema de control.*
- *Método de generación de tareas.*
- *Método de ejecución de tareas.*
- *Provisión de recuperación de errores en la ejecución de tareas y en otras fases de operación.*
- *Especificación de las descripciones de los recursos, su ubicación y provisiones para la manipulación de materiales.*

Por lo tanto, **una arquitectura de control define un modelo funcional genérico** cuyos *módulos tienen sus responsabilidades bien definidas, aunque manteniendo cierta autonomía en la forma de implementar dichas responsabilidades.* Además de *establecer la estructura del sistema* en su construcción, proporciona el *modo de dividir el problema* en partes resolubles, las cuales eventualmente se combinan en una solución completa.

Lo ideal sería disponer de una ***arquitectura de control universal*** *que aportase un elevado nivel de integración y estandarización. Esto supondría un incremento de la reutilización del software, porque se podrían construir herramientas de desarrollo o generadores de aplicación que simplificaran en gran medida el desarrollo de estos sistemas.*

1.3 Métodos estructurados de análisis y diseño.

El análisis y diseño estructurado surge a partir de los conceptos y técnicas de programación estructurada, y de las ideas tempranas sobre modularidad. Estos métodos o técnicas estructurados, denominados así genéricamente, *aplican construcciones de programación estructurada a las definiciones de procesos y datos.*

La modularidad es un aspecto clave, aunque la terminología y aplicación de este concepto sean distintas en análisis y diseño. No obstante, las guías para reconocer la independencia y simplicidad de la función son inherentes al enfoque */Marciniak, 94/.*

El análisis y diseño estructurado trabajan al nivel conceptual del sistema, con ayuda de un conjunto limitado de construcciones lógicas y una organización jerárquica planificada de la información. El modelo construido en el análisis ofrece la visión funcional del nuevo sistema, mientras que en diseño, se definen y organizan los detalles sobre la implementación planificada del sistema y el software diseñados.

En los apartados siguientes se presentan las metodologías más representativas dentro de este enfoque, indicando los elementos objeto de modelado, las herramientas empleadas y las fases identificadas en cada una de ellas. Seguidamente, se aporta un estudio de las metodologías descritas desde la perspectiva de la reutilización. Para finalizar, se describen aplicaciones que bien han empleado alguna de estas metodologías en el desarrollo del software de control de FMS, o han utilizado una nueva metodología inspirada en alguna de las aquí expuestas.

1.3.1 Principales métodos.

1.3.1.1 SA/SD (Structured Analysis/Structured Design).

La técnica de Tom DeMarco */DeMarco, 79/* es la más conocida en el ámbito del análisis estructurado, acompañada normalmente por la de Yourdon/Constantine */Yourdon, 79/* en lo referente al diseño.

Herramientas para construir los modelos de análisis según DeMarco

DeMarco utiliza para crear sus modelos de análisis los siguientes componentes:

- **Diagramas de flujo de datos (DFD - Data Flow Diagram).** Un DFD es una visión gráfica de los procesos de transformación, y sus interfaces o flujos. Su simbología se presenta en la figura 1.7.

- **Especificaciones de procesos (PSPEC – Process SPECification).** Una PSPEC es una descripción del proceso de transformación que realiza una burbuja en un DFD, indicando las restricciones impuestas al proceso, las características de rendimiento relevantes y las restricciones de diseño que pueden influir en la implementación de dicho proceso. Existen varios métodos alternativos a la descripción textual. Por citar algunos: lenguaje estructurado o pseudocódigo, tablas o árboles, ecuaciones matemáticas, diagramas o gráficos.

- **Diccionario de datos.** Recoge la definición, concisa y sin ambigüedad, de todos los flujos y almacenes referenciados por los procesos, incluyendo composición, valores, componentes de datos y cualquier relación entre los componentes.

En muchas aplicaciones es necesario adicionalmente presentar las relaciones entre complejas colecciones de datos. Para ello se ha de ampliar la notación e incluir un componente de modelado de datos: **diagramas de estructura de datos (DSD – Data Structure Diagram)** *[DeMarco, 79]* o **diagramas entidad/relación (E/R - Entity/Relationship diagram)** *[Topper, 94]*.

También son precisas ampliaciones en el caso de sistemas de tiempo real, los cuales *requieren la representación del flujo de control.* En este sentido, destacan las siguientes ampliaciones *[Pressman, 93]:*

- **Ward y Mellor.** Sus aportaciones consisten en *representar en el propio DFD* los siguientes aspectos:

  - *Información de control, su flujo, y el procesamiento de control asociado.*
  - *Instancias del mismo proceso (multitarea).*
  - *Flujos de información recogida o generada de forma continua.*

- *Estados del sistema y mecanismos que provocan estos cambios, descritos mediante diagramas transición estado (STD – State Transition Diagram).*

- **Hatley y Pirbhai.** Proponen una representación del control mediante:
  - *Diagramas de flujo de control (CFD – Control Flow Diagram).* Contiene los mismos procesos que el DFD, pero muestra el flujo de control. Los procesos de control no se describen directamente en el CFD, sino que se usa una referencia notacional (barra sólida) a una especificación de control.
  - *Especificación de control (CSPEC – Control SPECification).* Describe el comportamiento del software cuando se detecta un suceso o señal de control, y qué procesos se invocan como consecuencia de la ocurrencia del suceso. Para ello se emplean diversas herramientas: tablas de activación de procesos, tablas de decisión, diagramas de transición estado, etc.

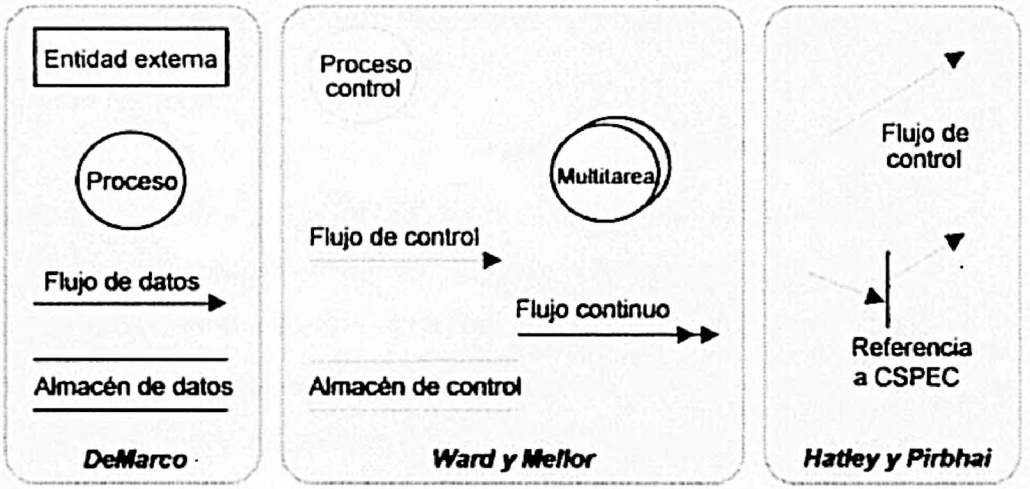


Figura 1.7: Notación de DeMarco y las ampliaciones para tiempo real.

Herramientas para construir los modelos de diseño por Yourdon y Constantine

Yourdon y Constantine emplean las siguientes herramientas:

- **Diagramas de estructura (SC – Structure Chart).** Herramienta gráfica para *representar la jerarquía de módulos.* Su *simbología* consta de: *rectángulos* para identificar los módulos, *flechas de conexión* para representar las llamadas entre

módulos, y *fechas de acoplamiento* para indicar el intercambio de información (datos o control) entre módulos.

Otras estructuras específicas que incluye la notación son: recursividad, iteración, estructuras con información oculta, referencias ocultas entre módulos, enlaces asíncronos, macros, rutinas de librerías, y condiciones de exclusión mutua.

- **Especificaciones de módulos (MSPEC – Module SPECification).** Una MSPEC es una descripción detallada de la tarea que realiza el módulo. Generalmente, se emplea pseudocódigo (inglés estructurado) o diagramas de acción.

Metodología

Conocidas las herramientas para realizar el modelado de datos y procesos en las etapas de análisis y diseño, los pasos de la metodología SA/SD (figura 1.8) se exponen a continuación:

- **Construcción del modelo de análisis.**
  - *Creación del modelo de flujo de datos.* El DFD0 (DFD de nivel 0) contiene la burbuja que representa el sistema, los flujos de datos compuestos y las entidades externas. La expansión del DFD0 al nivel 1 se realiza mediante un análisis gramatical: los nombres se corresponden con entidades externas, datos, control o almacenes; los verbos se asocian con procesos. El proceso de expansión o refinamiento continúa hasta conseguir que cada burbuja represente una función primitiva.
  - *Creación del modelo de flujo de control.* La revisión de la narrativa que describe el sistema permite seleccionar los elementos o sucesos de control. Dependiendo de la notación elegida, se incorporarán en los propios DFDs o se construirán los correspondientes DFCs.
  - *Desarrollo de las especificaciones.* Todos los procesos de los niveles inferiores del modelo del flujo de datos se describen en las PSPECs. Si es el caso, se desarrollan las CSPECs para especificar el comportamiento del sistema identificando sus estados y transiciones.

- *Validación del modelo de análisis.* Todos los componentes del modelo deben ser validados, prestando especial interés en comprobar que todos los flujos de un diagrama hijo estén representados en el diagrama padre por el mismo flujo o como parte de un flujo compuesto, y asociados a la burbuja adecuada.

- **Construcción del modelo de diseño.**
  - *Creación de los diagramas de estructura.* La conversión de los DFDs a SCs se realiza mediante las técnicas de Análisis de transformación y de transacción *[Yourdon, 79]*. Estas estrategias de diseño obtienen la estructura del sistema a partir del análisis del flujo de datos y transformación de los mismos (Análisis de transformación), o de las transacciones que debe procesar el sistema (Análisis de transacciones).
  - *Desarrollo de las especificaciones de módulos.* Los módulos identificados en los SCs se describen en las correspondientes MSPECs.

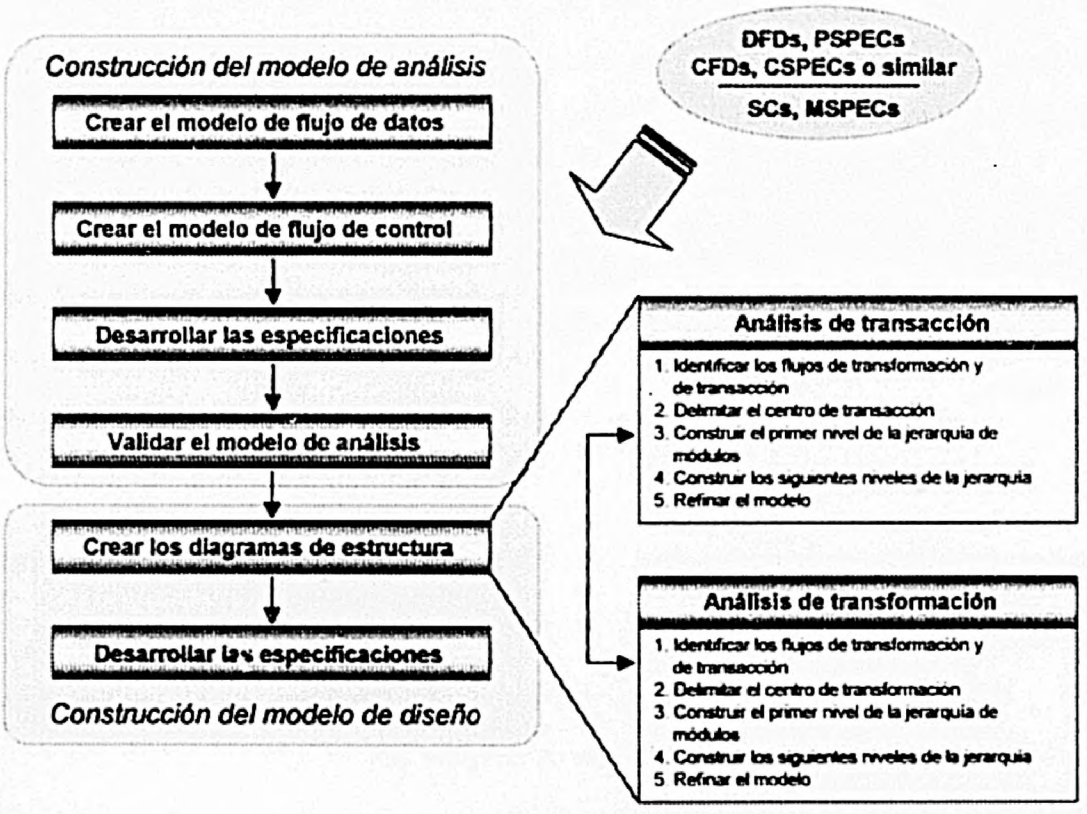


Figura 1.8: Metodología SA/SD.

Conclusión

La descomposición funcional característica del análisis, proporciona una jerarquía con una visión global del sistema en la parte superior (descrita mediante DFDs), *y primitivas o funciones de bajo nivel en la parte inferior* (descritas por PSPECs).

De forma similar, *el diseño de la solución consiste en construir una jerarquía de control, con un módulo en la parte superior que controla los módulos subsecuentes* (descritos todos ellos mediante SCs), *y aquellos módulos que no precisan mayor nivel de detalle en la parte inferior* (descritos mediante sus correspondientes MSPECs).

1.3.1.2 SADT (Structured Analysis and Design Technique).

SADT es una técnica desarrollada por Douglas T. Ross en 1.974, con la intención de proporcionar un *enfoque riguroso y disciplinado para comprender las necesidades de los usuarios antes de dar una solución de diseño.* Mulder et al. *[Mulder, 97]* recomiendan usar SADT en la planificación, análisis y diseño de arquitectura, y emplear otras técnicas para el diseño detallado.

Modelos

Un modelo SADT es una secuencia organizada de diagramas con sus correspondientes textos explicativos (concisos). Constituye una *representación gráfica de la estructura jerárquica del sistema,* revelando claramente las relaciones entre todos sus elementos. A medida que se avanza en la jerarquía, el modelo incorpora mayor nivel de detalle.

Existen dos tipos de modelos SADT; ambos contienen actividades y datos, pero atienden a diferente criterio de descomposición:

- **Modelo de actividades.** *Descripción de la descomposición funcional* del sistema. Su simbología se presenta en la figura 1.9.
- **Modelo de datos.** *Descripción de la descomposición de los datos.* Este modelo no es la imagen espejo del modelo de actividades, más bien se usa a modo de

diccionario de datos para proporcionar una definición más rigurosa de los mismos. Su simbología se presenta en la figura 1.9.

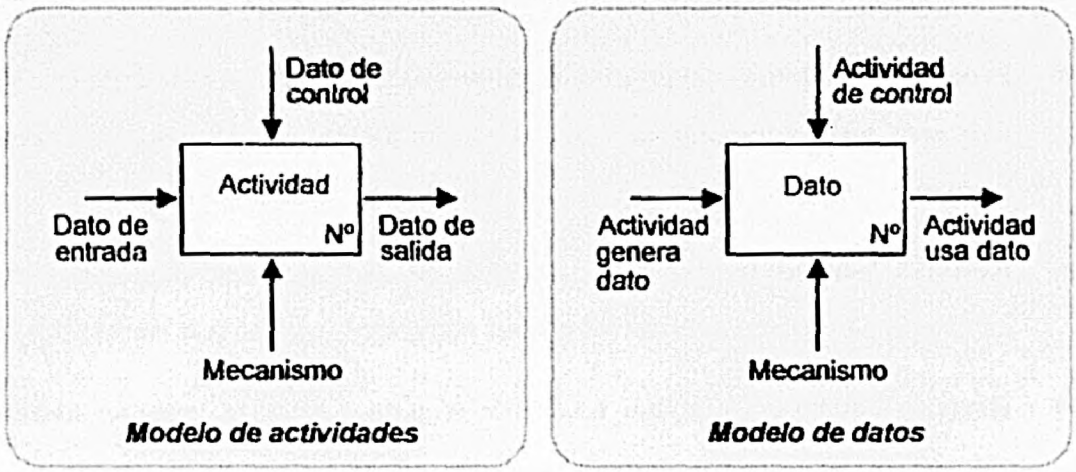


Figura 1.9: Simbología SADT.

Metodología

En la construcción de un modelo completo SADT es importante seguir un cierto procedimiento (figura 1.10). La metodología contempla **cuatro etapas** *[Mulder, 97]:*

- **Inicio del modelo.** El analista debe reunir y preparar la información, descomponer el problema y realizar una primera descomposición, según una serie de pasos:
  - *Determinar el propósito y punto de vista.* Plantear las preguntas que deben ser respondidas por el modelo que se va a construir.
  - *Generar la lista de datos.* Considerar los grupos y categorías principales de datos que el sistema genera y utiliza.
  - *Generar la lista de actividades.* Establecer las agregaciones de actividades que referencian cada una de las colecciones de datos identificadas.
  - *Dibujar el diagrama de alto nivel (diagrama A0).* Las listas de datos y actividades proporcionan el contenido de este diagrama, que consta de tres a seis cajas unidas por flechas.

- *Dibujar el diagrama de contexto (diagrama A-0).* Abstraer el diagrama A0 para presentar el sistema mediante una única caja con sus entradas, salidas, controles y mecanismos.

- **Proceso de modelado.** El objetivo es desarrollar un diagrama para cada caja del diagrama A0, y para cada caja de los nuevos diagramas. El proceso se repite hasta que el modelo describa el sistema con el nivel de detalle suficiente.

- **Revisión por parte del autor.** Construida la descomposición, se revisa exhaustivamente para detectar los errores cometidos (incluso con vuelta atrás).

- **Revisión externa.** El trabajo resultante se pasa a terceros para su revisión, obteniendo comentarios constructivos de mejora. Este punto es de gran importancia, ya que el éxito de SADT reside precisamente en la discusión entre analistas/diseñadores y terceros.

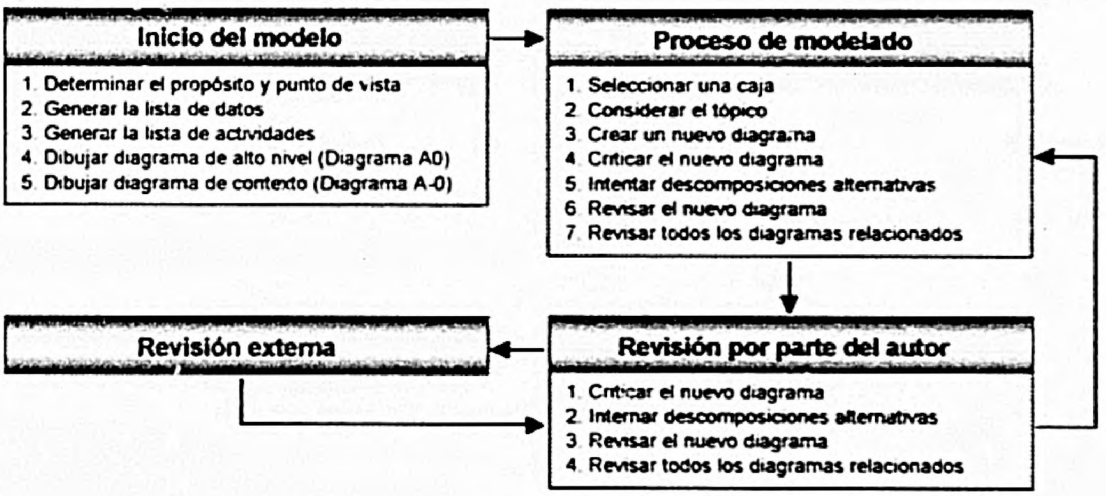


Figura 1.10: Metodología SADT.

Conclusión

A pesar de sus *carencias en términos de tiempo o espacio de almacenamiento*, esta metodología ha tenido *gran aceptación debido a su sencillez y simplicidad. Tampoco identifica las estructuras secuenciales, selectivas e iterativas* como tal, por lo que generalmente *no se emplea en la fase de diseño.*

1.3.1.3 IDEF0 (Integration DEFinition language 0).

IDEF0, basada en SADT, engloba la definición de un lenguaje de modelado gráfico (incluida su sintaxis y semántica) y la descripción de una metodología comprensiva para la construcción de modelos. Esta iniciativa del programa ICAM (Integrated Computer-Aided Manufacturing) estaba *orientada inicialmente al diseño de sistemas de fabricación avanzada.*

IDEF0 es una técnica de modelado que presenta una combinación de gráficos y texto de forma organizada y sistemática, con objeto de obtener una mejor comprensión del problema, apoyar el análisis, proporcionar una lógica de cara a modificaciones potenciales, especificar requisitos, y/o soportar las actividades de diseño e integración al nivel de sistema *[IDEF0, 93].*

Modelos

El modelo resultante consta de una jerarquía de diagramas, textos y glosarios con referencias cruzadas entre los mismos, que gradualmente muestran un mayor nivel de detalle. En los diagramas gráficos se recogen los componentes básicos de modelado: funciones (representadas por cajas), y los datos y elementos que relacionan dichas funciones (representados por flechas). Como apoyo a estos diagramas, se dispone de textos y glosarios que proporcionan información adicional.

Los **componentes del lenguaje** IDEF0, reflejo de los del modelo de actividades de SADT, se describen a continuación:

- **Cajas.** *Representan las funciones* definidas como actividades, procesos o transformaciones. Se organizan diagonalmente dentro del diagrama, siendo la posición indicadora de la importancia de dicha actividad en relación con las otras. Sus atributos se presentan en la figura 1.11.
- **Flechas.** *Representan las entradas, salidas, mecanismos y controles* que conectan las actividades. No identifican secuencias como en los modelos de

flujo de proceso tradicionales. Sus atributos son los siguientes: etiqueta, origen y destino.

- **Reglas.** Las reglas sintácticas y semánticas establecen el *modo de emplear e interpretar los componentes.*

- **Diagramas.** Proporcionan el *formato* para representar gráficamente y/o describir verbalmente los modelos. También aportan la base para realizar la gestión de configuración del modelo.

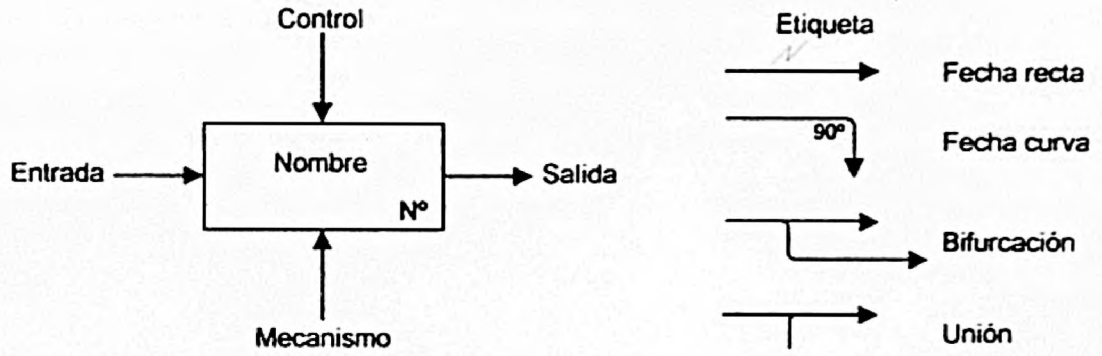


Figura 1.11: Sintaxis de las cajas y las flechas.

Metodología

IDEF0 prescribe los procedimientos para el desarrollo e interpretación de los modelos, incluyendo los correspondientes a la adquisición de datos, la construcción de diagramas, las revisiones y la documentación *[IDEF0, 93].* Pese a ello, algunos autores *[Wu, 94]* consideran que IDEF0 es una simple herramienta y no una metodología.

A continuación se describe el **procedimiento para la creación de modelos**, con gran similitud al correspondiente en la metodología SADT:

- **Determinar la orientación del modelo.** La guía para la creación del modelo reside en su orientación, la cual debe permanecer consistente, clara y sin distorsiones a lo largo de todo el proceso. Establecer la orientación *consiste en determinar: el contexto* (ámbito del modelo), *el propósito* (objetivos que se persiguen), *y el punto de vista* (aspectos que se enfatizan).

- **Construcción del diagrama de contexto.** El diagrama de contexto *(diagrama A-0)* presenta el contexto, propósito y punto de vista del modelo, y es objeto de los posteriores esfuerzos de descomposición. Este diagrama representa el sistema (única caja) y la relación con su entorno (flechas de entrada, control, mecanismo y salida).

- **Construcción del diagrama de alto nivel.** Realmente el modelo comienza con el *diagrama A0,* que descompone la función del diagrama A-0 en las funciones que la componen en el siguiente nivel de detalle (figura 1.12). El diagrama A0 presenta entre tres y seis cajas.

- **Construcción de los diagramas hijos.** Cada caja del diagrama A0 se descompone en sus funciones, teniendo la precaución de no perder información en dicha descomposición. Para obtener un diagrama claro, puede ser necesario separar o juntar funciones hasta que las funciones del diagrama A0 puedan expresarse entre tres y seis cajas. Aquellas partes que requieran clarificación, generarán diagramas con mayor nivel de detalle.

- **Elaboración del material de apoyo.** Eventualmente, cada diagrama dispone de información adicional en forma de:

  - *Texto.* Comentarios breves y concisos que no duplican la información contenida en el diagrama.
  - *Glosario.* Explicación de las definiciones asociadas con las funciones y datos/elementos.
  - *Diagrama FEO (For Exposition Only).* Descripción gráfica que destaca algún aspecto importante de un diagrama, pero sin seguir las normas IDEF0.

- **Selección de las cajas que requieren un mayor detalle.** La descomposición nivel a nivel es preferible a la descomposición en profundidad. No obstante, se aconseja que a la hora de decidir qué caja expandir, se comience con las menos familiares o menos claras, o con aquellas que proporcionen más información sobre otras funciones.

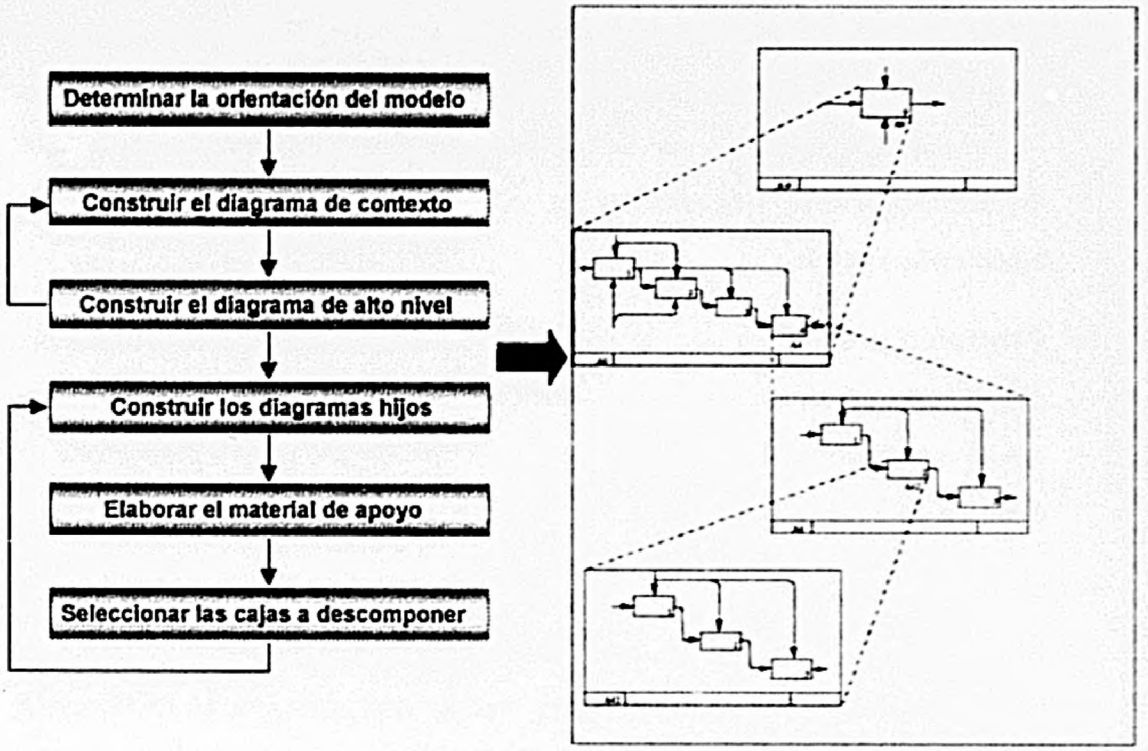


Figura 1.12: Metodología IDEF0.

Conclusión

IDEF0 proporciona una representación organizada de las actividades y las relaciones entre las mismas, pero sin tener en cuenta la variable tiempo. Tampoco soporta la especificación de los procesos o de los datos. Para cubrir estas deficiencias se dispone de otros métodos IDEF *[IDEF, 97]:* IDEF1X (modelado de datos) *[IDEF1X, 93]*, IDEF3 (descripción de los flujos de procesos) *[Mayer, 95]*.

1.3.1.4 MÉTRICA.

Desarrollada por el Ministerio para las Administraciones Públicas (MAP), MÉTRICA es una *guía formal, aunque flexible, para el diseño y construcción de sistemas de información.*

Aunque contempla aspectos de gestión de proyectos, gestión de calidad y gestión de configuración, MÉTRICA no pretende soportar todas las actividades relacionadas con estos conceptos. Sin embargo, *aporta el nexo de unión, identificando el lugar donde*

conectan la metodología de desarrollo de sistemas y los restantes aspectos asociados con la construcción de cualquier sistema de información.

Herramientas de modelado

Las principales técnicas para el modelado de procesos y de datos que usa MÉTRICA se presentan a continuación *[MÉTRICA, 95b]:*

- **Modelado de procesos:**
  - *Diagramas de flujo de datos (DFD).* Permite construir el modelo lógico del sistema (qué y no cómo lo hace), con independencia de la notación gráfica.
  - *Minieespecificaciones.* Bajo este término se agrupan el conjunto de técnicas para especificar los procesos: narrativa tradicional, lenguaje estructurado, tablas de decisión, árboles de decisión, etc.
  - *Diagramas de estructura (SC) o Diagramas de Estructura de Cuadros (DEC).* Representan la solución como una jerarquía de módulos, obtenida a partir de los DFDs aplicando las estrategias de diseño.
- **Modelado de datos:**
  - *Análisis de entidades (top-down).* Esta técnica permite identificar las entidades y sus relaciones, partiendo de una visión general para llegar al detalle. El objetivo es obtener el modelo entidad/relación (E/R) que constituye la representación conceptual de los datos.
  - *Normalización de datos (bottom-up).* Para mejorar o ampliar el modelo de datos, se aplica la normalización en tres pasos: primera, segunda y tercera forma normal.
  - *Historia de la Vida de las Entidades (HVE).* Diagrama que describe la evolución de las entidades de datos del sistema, los requisitos de tratamiento de las mismas, los estados posibles para que tengan lugar las transacciones, y los efectos de los eventos o sucesos en las entidades.
  - *Diccionario de datos.* Recoge información sobre todos los elementos que han ido surgiendo al aplicar las diferentes técnicas a lo largo de las distintas fases del ciclo de vida.

Metodología

A continuación se presentan las distintas **fases** identificadas en la metodología MÉTRICA versión 2.1 *[MÉTRICA, 95a]*, que también se ilustran en la figura 1.13.

- **Fase 0: Plan de sistemas de información.** Esta fase se omite en el desarrollo de sistema aislados; únicamente es precisa para la realización de un Plan de sistemas de información que asegure la adecuación entre los objetivos estratégicos de la organización y la información necesaria para soportarlos.

- **Fase 1: Análisis de sistemas.** El objetivo es obtener las especificaciones formales del sistema a desarrollar, que describan en detalle las necesidades de información y la arquitectura lógica del nuevo sistema (con independencia del entorno técnico). Esta fase se estructura en *dos módulos:*

  - *Análisis de Requisitos del Sistema (ARS).* Analizar y documentar las necesidades funcionales o de servicio del nuevo sistema.
  - *Especificación Funcional del Sistema (EFS).* Construir la especificación detallada del sistema, incluyendo los requisitos no funcionales (facilidades, restricciones de rendimiento, seguridad, etc.).

- **Fase 2: Diseño de sistemas.** Partiendo de las especificaciones funcionales, se realiza la descripción del sistema desde un punto de vista físico, considerando el entorno tecnológico específico en el que va a funcionar. Esta descripción incluye el diseño de la arquitectura del sistema y el esquema externo de los datos.

- **Fase 3: Construcción de sistemas.** Su objetivo es la construcción y prueba de los distintos componentes del sistema, a partir de sus especificaciones físicas. Con tal fin, se estructura esta fase en *dos módulos realizados en paralelo:*

  - *Desarrollo de Componentes del Sistema (DCS).* Preparar el entorno de construcción y pruebas, realizar la codificación, las pruebas unitarias y las pruebas de integración del software, desarrollar todos los procedimientos de operación de los componentes, y definir los recursos de formación.

- *Desarrollo de Procedimientos de Usuario (DPU).* Definir los procedimientos de usuario, la formación precisa para los mismos, y estimar los recursos de usuario necesarios para el entorno de trabajo.

- **Fase 4: Implantación de sistemas.** Construido el sistema, queda por realizar una comprobación exhaustiva total, haciendo énfasis en la verificación funcional, de rendimientos, y de condiciones extremas. Una vez concluidas las pruebas de sistema y de aceptación, se procede con las tareas de puesta en explotación del nuevo sistema.

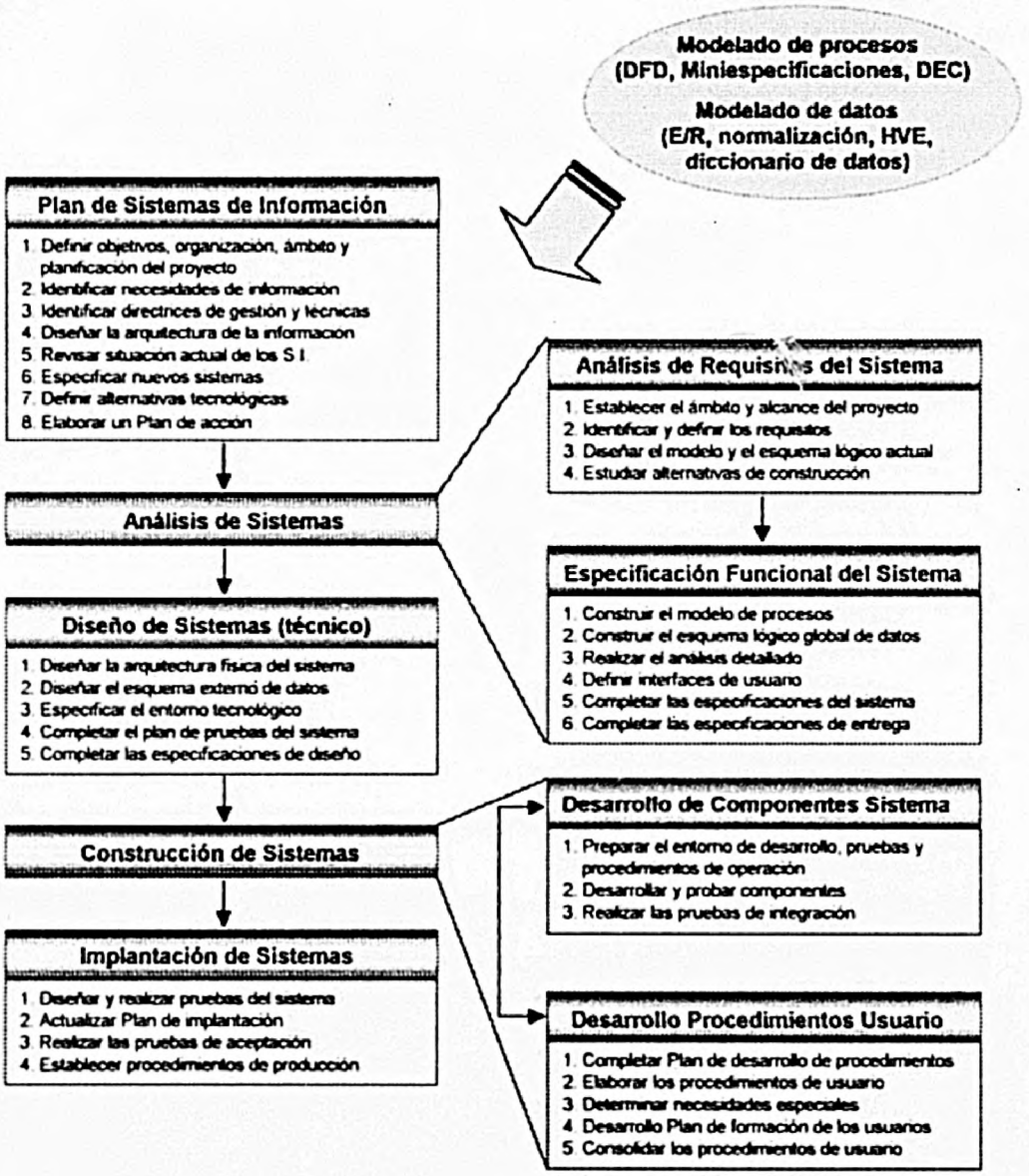


Figura 1.13: Metodología MÉTRICA.

Conclusión

MÉTRICA ofrece un marco de trabajo donde se *define: la estructura de proyecto guía del equipo de trabajo, los productos a desarrollar y las responsabilidades de los miembros del proyecto y usuarios, todo ello estructurado en una sucesión de fases, módulos, actividades y tareas.*

Esta metodología no está asociada al modelo de desarrollo de ciclo de vida en cascada, ya que *prescribe gran cantidad de retornos al nivel de actividades, módulos, e incluso de fases.* Además, *contempla la utilización de técnicas de prototipado y otras propias de desarrollos de tipo evolutivo e incremental.*

1.3.1.5 SSADM (Structured Systems Analysis Method).

SSADM es la *metodología para el desarrollo de sistemas de información,* utilizada por los departamentos del gobierno inglés. Dispone de unas actividades de soporte, denominadas procedimientos de proyecto (planificación de la capacidad, aseguramiento de la calidad, gestión de proyectos, evaluación de riesgos), que tienen interfaces con las actividades centrales, aunque los detalles sean ajenos al ámbito de esta metodología.

Modelos

SSADM proporciona **tres perspectivas del sistema**[2] que se pretende modelar:

- **Funcional.** Los DFDs presentan una visión secuencial de las funciones y los datos que fluyen entre las mismas, obviando la variable tiempo.
- **Estructural.** Las estructuras lógicas de datos (LDS - Logical Data Structure) garantizan la presencia de los datos, para que las funciones identificadas en los DFDs puedan realizar su tarea.

---

[2] Las técnicas empleadas en estos modelos han sido descritas en metodologías anteriores.

- **Con efectos de tiempo.** Las HVEs aseguran que los eventos ocurren en el tiempo, que se entienden, y que se modelan reglas integras para gobernarlos.

La primera vista se comprueba fre0nte a la segunda para confirmar que todos los datos se crean en algún momento. La tercera vista permite identificar invariablemente todos aquellos datos y/o aquellas funciones que han sido omitidas.

Metodología

La metodología SSADM **se estructura en módulos, y éstos en etapas.** Para cada etapa, prescribe los productos que hay que generar, cuando generarlos y cómo deben generarse. La estructura de la metodología SSADM versión 4 */Barn, 97/* se presenta a continuación:

- **Estudio de viabilidad.** Este estudio pretende definir el ámbito del proyecto y la dirección que tomará. La construcción de los DFDs y LDSs del sistema ayudarán a escudriñar estos problemas.

- **Análisis de requisitos.** Los modelos funcional y estructural del sistema proporcionan: las funciones del entorno actual, el modelo de datos que las soporta, y los problemas y requisitos a acometer. Las soluciones alternativas resultantes se presentan al usuario.

- **Especificación de requisitos.** Elegida la opción, se desarrolla su especificación funcional detallada; se amplía el conocimiento de los requisitos mediante HVEs, análisis de datos relacionales (RDA – Relational Data Analysis), y sesiones de revisión de prototipos con el usuario.

- **Especificación lógica del sistema.** El usuario debe decidir el entorno técnico en el que operará el nuevo sistema, aunque se construya un diseño lógico (independiente del entorno) que especifica la interface de usuario y los procesos de actualización y consulta a la base de datos.

- **Diseño físico.** La especificación lógica del sistema se traduce en la especificación de los programas y el diseño físico de datos, teniendo presente el

entorno de desarrollo elegido, las guías de productos de los suministradores, y los propios estándares de la instalación.

Conclusión

SSADM no cubre todos los aspectos del ciclo de vida de un proyecto de desarrollo: *la planificación de sistemas de información estratégicos está fuera de su alcance; el diseño resultante es genérico y precisa traducción al entorno concreto de construcción; la documentación generada no abarca la fase de mantenimiento.*

SSADM guía al diseñador en el desarrollo del software, el procesamiento de datos y los tipos de almacenamiento de datos. Pero para uso industrial, precisa mayor estructuración de las últimas fases, dado el impacto de éstas en el éxito del proyecto.

1.3.1.6 MERISE.

MERISE *[Waldner, 92]* es un *método de análisis y diseño para sistemas de información*, muy difundido en Francia, que recomienda la división del problema en una jerarquía de niveles, con objeto de obtener una descripción exhaustiva de los datos y procesos involucrados en la aplicación.

Modelos

El resultado del análisis funcional se expresa por medio de los **modelos conceptuales, los modelos lógicos y los modelos físicos** tanto de datos como de procesos.

- **Nivel conceptual.** La formalización a este nivel representa los datos y tratamientos necesarios en el sistema de información, en respuesta a ¿Qué?
- **Nivel lógica.** Este nivel añade las nociones de tiempo, lugar y responsable a la formalización conceptual, es decir, responde a ¿Dónde?, ¿Cuándo? y ¿Quién?
- **Nivel física.** Sobre la base de las necesidades planteadas en los niveles previos se define la solución técnica del sistema final, respondiendo a ¿Cómo?

Metodología

La construcción de un sistema de información requiere los siguientes pasos:

- **Creación del modelo conceptual.** Formalización de las especificaciones de usuarios y diseñadores mediante una representación común, la cual consta de:
  - *Modelo conceptual de datos.* Identifica los datos y las dependencias funcionales entre los mismos.
  - *Modelo conceptual de procesos.* Formalización de los procesos, para lo cual: se identifican todos los eventos[3], se especifican los procesos, y se definen las condiciones de disparo y sincronización de dichos procesos.
- **Creación del modelo lógico.** El modelo conceptual se reestructura en una representación teórica válida para cualquier sistema de procesamiento sin restricciones en cuanto a tipos o productos particulares. Se distinguen:
  - *Modelo lógico de datos.* Especifica las responsabilidades de creación y actualización de los datos, su localización geográfica, sus puntos de acceso, e incluso, los aspectos funcionales de las redes de comunicaciones.
  - *Modelo lógico de procesos.* Los eventos y operaciones del modelo conceptual de procesos se transforman en flujos de información y procedimientos funcionales.
- **Creación del modelo físico.** Validado el modelo lógico y conocido el entorno hardware/software, el modelo físico propone la estrategia de producción del software para su posterior codificación. Engloba dos modelos:
  - *Modelo físico de datos.* Su objetivo es la optimización de la gestión de los datos según la herramienta elegida y los procesos que empleen dichos datos.
  - *Modelo físico de procesos.* Este modelo contempla dos formalizaciones: el modelo físico externo, que especifica las características de hardware, redes,

---

[3] Un proceso está siempre relacionado con un evento.

impresoras, etc. así como el número de máquinas, y el modelo físico interno, que presenta la ejecución de los procesos y el software de base.

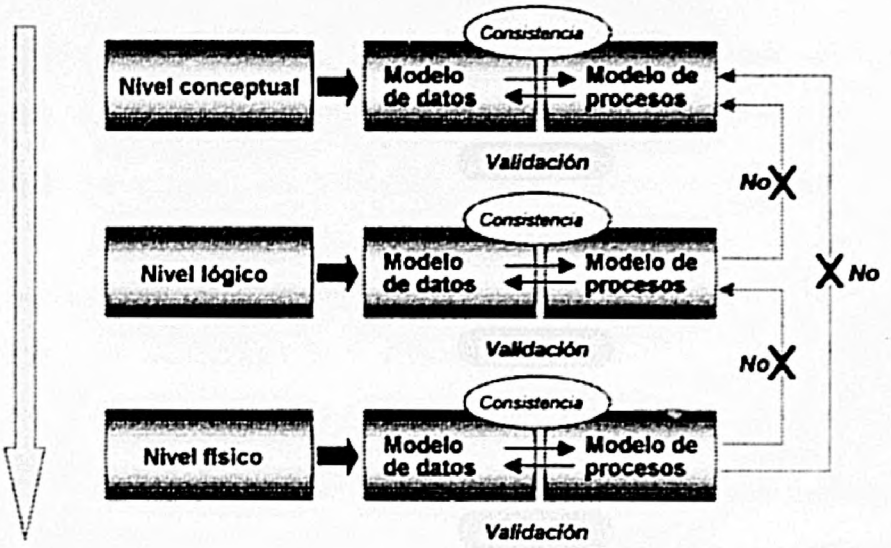


Figura 1.14: Metodología MERISE.

Conclusión

MERISE presenta un procedimiento analítico que evoluciona de lo general a lo particular, *estudiando el dominio completo antes de considerar los subsistemas* en detalle. Además, propone el análisis por separado de datos y procesos en cada nivel, teniendo *especial cuidado en la comprobación de la consistencia de ambos análisis antes de la validación y paso al nivel siguiente.*

1.3.2 Análisis de los métodos estructurados desde la perspectiva de la reutilización.

Las metodologías estructuradas descritas en los apartados anteriores (SA/SD, SADT, IDEF0, MÉTRICA, SSADM y MERISE) **consideran la reutilización de forma implícita;** todas ellas abogan por la construcción de modelos modulares independientes de la solución técnica concreta para la aplicación particular, salvo en las últimas fases de diseño. Pero esto no es suficiente sin una mentalidad abierta a la reutilización, aún más cuando estos métodos *no disponen de mecanismos potentes que simplifiquen esta tarea.*

Las **construcciones básicas** de los métodos estructurados *son los procesos y los datos.* La mayoría de las metodologías disponen de estos conceptos así como de las correspondientes **notaciones gráficas** para representarlos. La excepción se encuentra en *IDEF0, que únicamente contempla las funciones y su representación.*

En el caso de sistemas de tiempo real también es necesario construir modelos del comportamiento de la aplicación. En este sentido destacan las ampliaciones de Ward & Mellor y Hatley & Pirbhai a SA/SD, que representan la información de control, su flujo y el procesamiento asociado. Aunque de forma menos exhaustiva, también se contempla esta necesidad en MÉTRICA, SSADM y MERISE, mientras que *SADT e IDEF0 únicamente proporcionan modelos estáticos de la aplicación.*

Los **procesos** de desarrollo propuestos por las distintas metodologías *no incluyen la reutilización explícitamente.* Sin embargo, consideran la reutilización de forma implícita, contemplan el desarrollo con reutilización de funciones y módulos, tratan el desarrollo para reutilizar a varios niveles (funciones, módulos, subsistemas, dominios) de forma implícita, y no presentan conatos de gestión pese a la existencia de librerías de funciones.

A continuación, **se analizan con mayor detalle los distintos procesos,** indicando únicamente aquellos aspectos que afectan a la reutilización:

Análisis de SA/SD

Una característica deseable en todo modelo de análisis es que *sus componentes sean ortogonales;* es decir, que los componentes no se solapen y que no exista redundancia en el modelo. Además, esta metodología emplea el término lógico para describir un *modelo que define un dominio* o sistema con independencia de cualquier consideración tecnológica u operativa, de modo que *pueda ser reutilizado para crear distintas arquitecturas del sistema o diseños software.*

También *establece los criterios para un buen diseño:*

- *El tamaño del módulo debe ser tan pequeño como sea posible.*

- *El propósito del módulo debe ser específico.* Esto significa maximizar la cohesión[4].
- *Las referencias a los datos deben ser precisas.* Esto significa minimizar el acoplamiento[5].

Estas sugerencias de diseño persiguen la *independencia de los módulos*, y las correspondientes repercusiones: *se reduce el número de módulos afectados ante un cambio en los requisitos, se facilita la lectura y compresión de los diagramas, y se obtienen módulos con un mayor potencial de reutilización.*

Análisis de SADT / IDEF0

SADT e IDEF0 proporcionan una sistematización efectiva para definir las actividades del modelado funcional y refinarlas hasta el nivel de detalle que sea preciso. El resultado es la *especificación del dominio representada en forma de modelos (y material de apoyo) más o menos exhaustivos que pueden ser reutilizados.*

Análisis de MÉTRICA

Uno de los objetivos de esta metodología es el desarrollo de sistemas que satisfagan las necesidades de información de los usuarios, *teniendo presente el posterior mantenimiento de la aplicación, y la incorporación de modificaciones o ampliaciones futuras.*

El análisis describe el funcionamiento del sistema de modo conceptual, es decir, *eliminando todas las referencias al entorno físico.* También especifica los subsistemas principales y sus componentes.

El diseño *identifica las actividades físicas con características comunes en cuanto al proceso, con objeto de disminuir el esfuerzo de implementación por medio de la*

---

[4] La cohesión es la medida de la fortaleza de la asociación entre los elementos que componen un módulo.

[5] El acoplamiento es la medida de la interdependencia entre los módulos.

reutilización del código. Asimismo, determina aquellas actividades existentes en las *librerías generales de desarrollo* a modo de rutinas de uso general.

En el diseño de nuevos componentes *recomienda* realizar el refinamiento de los mismos con el fin de reducir la complejidad de las interfaces (*disminuir el acoplamiento* entre módulos) e incrementar la consistencia de las funciones desempeñadas por cada módulo (*aumentar la cohesión* de los módulos).

Para la generación de código, *MÉTRICA recomienda el empleo de estándares de programación que simplifiquen las tareas de mantenimiento.* También contempla la *introducción de lenguajes de cuarta generación y de técnicas de Ingeniería del software que automaticen este proceso de construcción.*

Análisis de SSADM

La propia concepción de SSADM expresa explícitamente la necesidad de *asegurar la independencia respecto al suministrador,* y por ende, evitar trabarse en soluciones propietarias. La especificación de requisitos no considera ningún hardware ni software específico, y *el diseño lógico puede implementarse en distintas plataformas* sin necesidad de modificaciones o ajustes.

SSADM está publicado como un estándar abierto, que manifiesta que los suministradores deberían proporcionar las guías específicas del producto para traducir los diseños lógicos en implementaciones específicas de dicho producto.

Análisis de MERISE

MERISE propugna el análisis de datos y procesos en tres niveles de abstracción, y únicamente en el último (nivel físico) realiza la formalización teniendo en cuenta la solución técnica concreta. Por lo tanto, *las formalizaciones correspondientes al nivel conceptual y al nivel lógico son potencialmente reutilizables.*

Además, *también permite la reutilización al nivel de dominio y de subsistema,* ya que el propio procedimiento analítico evoluciona desde el estudio general del dominio completo hasta el análisis detallado de los distintos subsistemas.

Conclusión

Ninguna de estas metodologías establece procesos explícitos que soporten la reutilización. El esfuerzo necesario para conseguirlo es mayor que en el caso de otras tecnologías, debido a la *ausencia de mecanismos que fomenten la reutilización.*

No obstante, un primer paso en este sentido es la idea de Topper et al. *[Topper, 94]*, que proponen **la reutilización como un buen criterio para llevar a cabo la descomposición funcional** de un sistema, obteniendo módulos con un alto potencial para la reutilización.

Un indicador de este potencial es el tamaño, debido a que es más difícil reutilizar grandes programas que aquellos más pequeños. Por supuesto, pequeños módulos independientes proporcionarán poca funcionalidad. Buen ejemplo de ello se encuentra en las librerías de funciones matemáticas: los módulos con una única tarea y con una gran cohesión son más fáciles de reutilizar. Por lo tanto, *existe una convergencia de todos los criterios de cohesión, reutilización y esfuerzo hacia módulos pequeños con funcionalidad única.*

1.3.3 Aplicación de los métodos estructurados en el desarrollo del software de control de FMSs.

A continuación se presentan algunas aplicaciones de control para FMSs desarrolladas con métodos estructurados. *Ninguna tiene presente la reutilización explícitamente, aunque en algún caso ésta se consiga mediante generadores de aplicación.*

Los métodos estructurados más empleados en el desarrollo de software de control de FMSs son SADT e IDEF0. Pocas referencias se encuentran de SA/SD, SSADM y MERISE, aunque se comenten como otros métodos de desarrollo. De MÉTRICA no se conocen referencias, quizás por su estrecha vinculación con el área de gestión.

1. Método de diseño genérico desarrollado por Ranky

Ranky *[Ranky, 90]* desarrolla el modelo de un método de diseño genérico, que establece los pasos para construir un FMS concebido con la filosofía CIM, incluido su

software de control. *Los modelos estáticos obtenidos con SADT/IDEF0* aportan la especificación, definición y planificación del sistema, y constituyen la base sobre la cual se construyen los modelos dinámicos. Estos últimos se obtienen por medio de técnicas de modelado de evento discreto, modelado sólido y simulación, redes de Petri u otras, y su cometido es simular el rendimiento operativo del futuro sistema.

Aunque no cite explícitamente el término reutilización, Ranky lo tiene presente cuando afirma:

> *"La metodología es comprensible tanto por humanos como por máquinas, por lo tanto, en el futuro, los ordenadores podrían traducir los modelos directamente en código de control del sistema, diseños de estructuras de bases de datos, etc.".*

2. Modelo conceptual de un sistema de fabricación bajo pedido desarrollado por Wu

Wu *[Wu, 92] emplea IDEF0* para desarrollar el modelo conceptual de un sistema de fabricación bajo pedido (OHMS – Order Handling Manufacturing System). Un sistema de estas características involucra desde operaciones de producción bajo pedido hasta producto final adaptado al cliente, según una secuencia de actividades que afectan a casi todos los aspectos de los procesos de fabricación secundarios.

Wu indica que IDEF0 es una buena técnica para especificar la estructura de la información y organización de un sistema de fabricación complejo como OHMS, aunque es consciente de sus deficiencias. No tiene presente la reutilización, pero recuerda que IDEF0 es sólo una técnica y que la calidad de los modelos reside en la habilidad de los analistas. Por lo tanto, *el potencial de reutilización de dichos modelos depende de las pretensiones de las personas que los lleven a cabo.*

3. Metodología de diseño y evaluación propuesta por Wu

Wu considera que *los métodos estructurados existentes se centran en el diseño y se olvidan de la evaluación.* Por ello, propone una metodología de diseño y evaluación de sistemas de fabricación, que se estructura en las siguientes etapas:

- **Análisis de la situación.** Genera un modelo del sistema de fabricación existente (si es caso) y una lista con las demandas actuales. El procedimiento para realizar el análisis se denomina 5W (What, Why, Whether, Which and What):
  - *What.* Análisis detallado de la tecnología de producción, de la infraestructura de personal, de los productos y de los procesos.
  - *Why & Whether.* Constatación de la construcción o mejora del sistema.
  - *Which & What.* Identificación de los subsistemas, describiendo los sistemas físicos y de control, y realizando tanto análisis estático como dinámico.
- **Establecimiento de los objetivos.** Mediante análisis de entrada/salida (análisis I/O), se determinan los objetivos, los cuales definen la situación futura y por ende, la infraestructura de la planta a largo plazo.
- **Modelado conceptual.** Considera los siguientes pasos:
  - *Identificación de requisitos funcionales.* Se obtienen comparando las funciones del futuro sistema con las ya existentes. Los nuevos requisitos quedan sujetos a la decisión estratégica de desarrollarlos o adquirirlos.
  - *Organización de las funciones.* Para ello se emplea una herramienta de descomposición funcional[6]: DFD, IDEF0, SADT, redes GRAI o diagramas I/O (IOD – I/O Diagram). Normalmente, se aplica la descomposición de IODs hasta el nivel de componente para obtener los distintos modelos del sistema de fabricación.
  - *Análisis de los sistemas de control.* Las funciones de control se incorporan al modelo convirtiendo el IOD en un DFD o un modelo IDEF0. La descomposición de estas funciones se realiza con SSADM.
- **Diseño detallado.** El modelo conceptual se traduce en especificaciones detalladas que pueden ser implementadas, según los siguientes pasos:

---

[6] En una edición posterior *[Wu, 94]*, también incorpora su método orientado a objetos HOOMA.

- *Selección de la tecnología de producción.* En algunos casos es suficiente con la tecnología de producción existente, otras máquinas deben ser reemplazadas, y en otros casos, se requiere adquirir dicha tecnología.
- *Organización y layout de la tecnología de producción.* La disposición se realiza de acuerdo a los siguientes criterios: minimizar costes de transporte y manipulación, minimizar congestiones y retrasos, y maximizar la utilización del espacio, trabajos y recursos.
- *Desarrollo del sistema de información.* Incluye el diseño de los procesos y bases de datos, la selección y ubicación del hardware, y la especificación de las responsabilidades. Los métodos recomendados son los siguientes: SSADM para el desarrollo de las bases de datos, DFDs para el análisis del flujo de información, y diagramas E/R para las estructuras de datos relacionales.

- **Evaluación y decisión.** Las fases de evaluación y decisión tienen lugar al finalizar las etapas de modelado conceptual y diseño detallado: selección de la opción tecnológica y aceptación del diseño detallado, respectivamente.

Esta metodología se ha aplicado en la División de actuación de la empresa Lucas en Fordhouses (UK), cuyo cometido es el diseño y fabricación de sistemas de actuación hidráulicos avanzados para la aviónica militar y civil.

Desde luego **la reutilización no es parte integrante de este enfoque.** No obstante, en la etapa de "Establecimiento de objetivos", Wu indica que tiene gran importancia establecer los objetivos haciendo balance entre las necesidades de proyectos individuales y los objetivos a largo plazo especificados en la estrategia corporativa.

4. Arquitectura funcional de control descrita por Bauer

Bauer et al. *[Bauer, 94] emplean SADT como técnica de modelado* para describir su arquitectura funcional de control. El propósito es obtener una perspectiva que adopte un enfoque de análisis top-down: comienzan con una descripción completa del contexto de un sistema de control de planta, que se va descomponiendo gradualmente para describir

las actividades correspondientes al controlador de las actividades de producción PAC y al controlador de nivel superior FC.

Estos autores consideran que el mejor modo de implementar su trabajo es con un entorno de las tecnologías de la información, que combine el potencial de procesamiento de los ordenadores con el talento creativo de las personas. Por lo tanto, **la reutilización se efectúa con la herramienta de diseño AG (Application Generator).**

Esta herramienta se ha empleado en la implementación de dos aplicaciones industriales: una en el área de fabricación de periféricos de Digital Equipment Corporation en Clonmel (Irlanda), y otra en un FMS de Combo en Turín (Italia).

5. Método de diseño de sistemas de información de planta propuesto por Gong

Gong *[Gong, 96]* propone un enfoque estructurado para el diseño de sistemas de información de planta, que *representa conjuntamente la información, el control y la comunicación. A partir de estos modelos se obtienen los diagramas de tiempo para el paso de mensajes, característicos de un enfoque orientado a objetos.*

Con IDEF0 se pueden formalizar los flujos de materiales y de información explícitamente, pero no el paso de mensajes. Para incorporar este aspecto, Gong reemplaza las entradas y salidas originales por tres tipos de líneas: una línea gruesa indica el flujo de materiales, una línea delgada identifica el flujo de información y una línea punteada denota el paso de mensajes.

Este enfoque para el diseño de sistemas de información de planta incluye los siguientes **pasos:**

- **Construcción de un modelo IDEF0 a partir del flujo de producción de piezas.**
- **Representación del flujo de materiales** mediante líneas gruesas.
- **Representación del flujo de información** mediante líneas delgadas.

- **Refinamiento del modelo** por descomposición sucesiva hasta un nivel en el que cada bloque contenga un tipo de objeto agente (pieza, productor, distribuidor, monitor o base de datos).
- **Identificación del paso de mensajes de cada objeto agente, y construcción del correspondiente diagrama de tiempo.**

Aunque emplee la tecnología orientada a objetos y destaque la encapsulación, los **aspectos relativos a la reutilización se obvian totalmente,** ni siquiera se presenta como beneficio potencial.

6. Metodología GRAI-GIM

Otras metodologías combinan distintos métodos estructurados para el desarrollo de los sistemas de control. Este es el caso de **GRAI-GIM** *[Chen, 96]* que *aplica IDEF0 para realizar los modelos físico y organizativo, y MERISE para construir el modelo de información.*

1.4 Tecnología orientada a objetos.

El fundamento de este enfoque es la encapsulación. Un objeto encapsula datos, operaciones, otros objetos, constantes e información relacionada *[Pressman, 93]*. Esto significa que toda esta información está empaquetada con un único nombre.

Su contribución consiste en restringir los efectos de los cambios, puesto que todo acceso a los datos está gestionado mediante procedimientos, estableciendo una separación clara entre los aspectos externos del objeto (accesibles por otros objetos) y los detalles de implementación del mismo (ocultos para otros objetos).

La herencia es la parte más innovadora del enfoque, que permite retransmitir automáticamente el código a nuevas clases que asumen las características de las superclases genéricas, aunque con atributos y métodos propios.

Con relación a la programación estructurada, incorpora nuevos elementos: clases, objetos, y herencia como soporte de la nueva tecnología; abstracción, encapsulación, sobrecarga, polimorfismo o reutilización, como cualidades del producto obtenido.

Los aspectos generales de los métodos orientados a objetos, así como los modelos, las técnicas de modelado y la metodología que proponen los métodos más representativos, se describen en los apartados siguientes. También se aporta un estudio de los mismos desde la perspectiva de la reutilización, y algunas aplicaciones de esta tecnología al desarrollo del software de control de FMSs.

1.4.1 Aspectos generales de los métodos orientados a objetos.

Con independencia de la metodología y notación, **existe consenso** en las ideas centrales sobre la aplicación de la tecnología orientada a objetos para el desarrollo de grandes proyectos *[Corriveau, 96]:*

- **El proceso de desarrollo es iterativo e incremental.**
- **El análisis y diseño precisan tres perspectivas interrelacionadas del sistema:**

- *Modelado de la estructura.* Presenta la estructura estática del sistema, identificando las clases de objetos y sus relaciones.
- *Modelado del comportamiento.* Recoge el comportamiento del sistema y de sus componentes en función del tiempo.
- *Modelado de las relaciones.* Refleja los procesos funcionales de los objetos y la dependencia funcional entre objetos, obviando la variable tiempo.

El análisis comienza con una definición del problema, acuñada entre cliente y desarrollador. Esta definición inicial se transforma posteriormente en unos modelos, que conjuntamente constituyen una *representación precisa y concisa del problema.*

Durante el diseño, *estos modelos se refinan con objeto de construir una solución técnica* en forma de arquitectura del sistema, la cual establece la estrategia global de implementación. La tarea de refinamiento y optimización de dichos modelos continúa hasta obtener los esquemas detallados de programación.

Las diferencias entre los distintos métodos enmarcados en este enfoque estriban en:

- *El énfasis que hacen en los modelos* (cual de ellos es el más importante).
- *La notación particular que puede añadir información adicional a los modelos.*
- *Los pasos recomendados para el desarrollo de dichos modelos.*

1.4.2 Principales métodos.

1.4.2.1 OOADA (Object-Oriented Analysis and Design).

OOADA */Booch, 94/* considera el estudio del sistema desde múltiples perspectivas: la *estructura lógica*, en términos de clases y objetos, la *estructura física*, en términos de módulos y procesos, y sus correspondientes *aspectos estáticos y dinámicos.*

Técnicas de modelado

OOADA dispone de **cuatro diagramas básicos** (clases, objetos, módulos y procesos) y **dos diagramas suplementarios** (transición estado e interacción), que permiten describir el sistema desde las perspectivas previamente comentadas:

- **Diagramas de objetos.** Representan los objetos existentes y las relaciones estructurales entre los mismos (diseño lógico del sistema).
- **Diagramas de clases.** Describen las clases existentes y las relaciones entre las mismas (diseño lógico del sistema). Su notación se ilustra en la figura 1.15.
- **Diagramas de transición estado.** Especifican los estados de una instancia de una clase, los eventos que causan la transición de un estado a otro, y las acciones resultantes del cambio de estado (aspectos dinámicos).
- **Diagramas de interacción.** Trazan la ejecución de un escenario en el mismo contexto que un diagrama de objetos, identificando el paso de mensajes con un orden relativo (aspectos dinámicos).
- **Diagramas de módulos.** Presentan la agrupación de clases y objetos en módulos (diseño físico del sistema).
- **Diagramas de procesos.** Describen la distribución de los procesos a los procesadores (diseño físico del sistema).

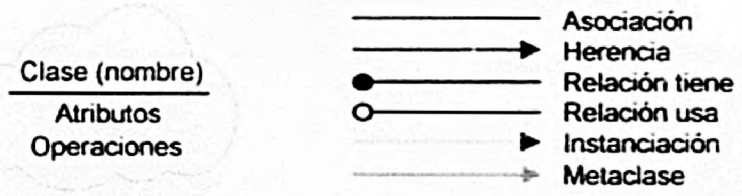


Figura 1.15: Notación básica del diagrama de clases OOADA.

Metodología

Booch propone un *ciclo de vida evolutivo a través del cual se refinan las vistas lógica y física del sistema.* Para ello estructura este ciclo en **dos procesos:** micro proceso y macro proceso (figura 1.16).

- **Micro proceso.** *Engloba las tareas diarias del desarollador.* Las actividades que abarca son las siguientes:
  - *Identificación de las clases y objetos con cierto nivel de abstracción.* Establecer el ámbito del problema. La descomposición del sistema requiere la búsqueda de las clases y objetos significativos, y de los mecanismos que proporcionan el comportamiento deseado a los mismos.
  - *Identificación de la semántica de las clases y objetos.* Refinar las abstracciones seleccionadas, estableciendo sus atributos y comportamientos, y distribuyendo responsabilidades. En esta etapa también se identifican los patrones de comportamiento comunes.
  - *Identificación de las relaciones entre las clases y objetos.* Asentar el ámbito de cada abstracción, y especificar la cooperación entre clases y objetos.
  - *Especificación de las interfaces y posterior implementación de las clases y objetos.* Seleccionar las estructuras y los algoritmos de las abstracciones, obteniendo el modelo físico del sistema. También se añaden nuevas clases y objetos, y se refinan los existentes de cara a su posterior implementación.

- **Macro proceso.** *Controla al micro proceso, y referencia las actividades del equipo de desarrollo completo en una escala de tiempo de semanas o meses.* Comprende las siguientes etapas:
  - *Concepción.* Establecer los requisitos del software y validarlos. Normalmente, la validación se realiza mediante prototipos.
  - *Análisis.* Describir el problema, que precisa la identificación de las clases y objetos del dominio, haciendo énfasis en el comportamiento del sistema (funciones, ciclos de vida, máquinas de estado de las clases y escenarios).
  - *Diseño.* Construir una arquitectura de la cual se derive la implementación, y establecer los aspectos tácticos de los distintos elementos del sistema.
  - *Evolución.* El software final se obtiene como resultado del crecimiento y modificación de la implementación a través de un refinamiento sucesivo.

- *Mantenimiento.* Evolución del sistema después de la entrega, que presentan la mayoría de los cambios en forma de nuevos requisitos o de eliminación de fallos.

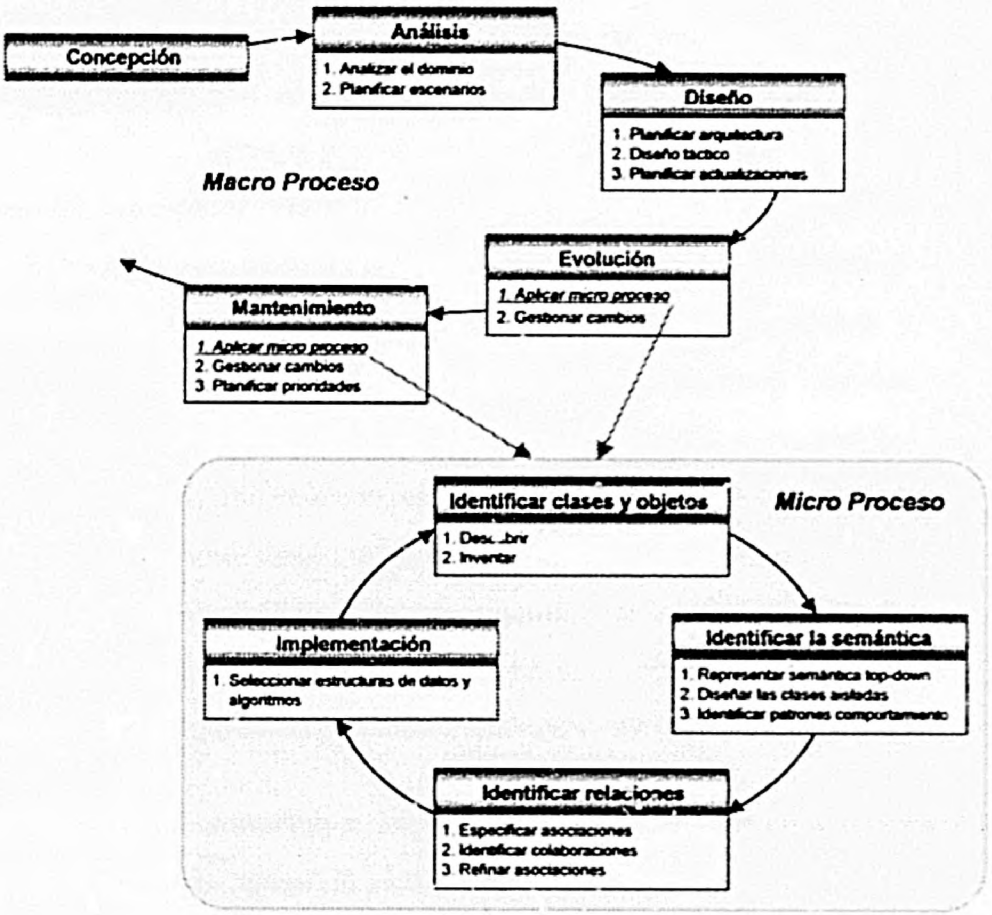


Figura 1.16: Metodología OOADA.

Conclusión

OOADA responde a un ciclo de vida evolutivo. La aplicación del micro proceso en las etapas de evolución y mantenimiento comienza con el análisis de requisitos de la siguiente actualización, tras lo cual se diseña una arquitectura, y se determinan las clases y objetos necesarios para implementar dicho diseño.

El resultado es un conjunto de actualizaciones ejecutables que representan refinamientos sucesivos de la primera versión de la arquitectura. También se construyen prototipos para explorar alternativas de diseño o investigar funcionalidades desconocidas o confusas.

1.4.2.2 OL (Object Lifecycles).

OL *[Shlaer, 92]* es una completa revisión del método original OOSA (Object-Oriented Systems Analysis) *[Shlaer, 88]*. OOSA se centra en la identificación, formalización y verificación del conocimiento como medio para determinar los requisitos de información del sistema, mientras que *OL modela el comportamiento dinámico del sistema en términos de modelos de procesos y estados.*

OL propone una división inicial del sistema en dominios[7]*, y de éstos en subsistemas*[8]. Cada división se analiza en tres pasos: modelado de la información, de los estados y de los procesos. *El diseño se deriva de los modelos de análisis de forma directa, y se representa mediante la notación gráfica y textual independiente del lenguaje OODLE (Object-Oriented Design LanguagE).*

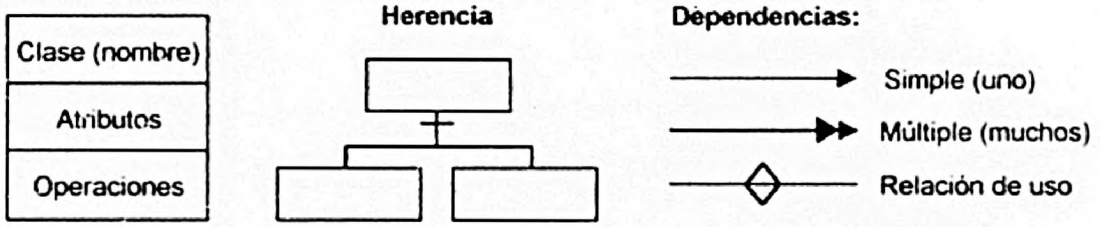


Figura 1.17: Notación básica OODLE.

Técnicas de modelado

Las técnicas de modelado que emplea OL se describen brevemente a continuación:

- **Fase de análisis.**
  - *Diagramas entidad relación (E/R).* También conocidos como diagramas de estructura de información o simplemente modelo de información, permiten modelar la información.

---

[7] "Un dominio es un mundo aislado real, hipotético o abstracto ocupado por un conjunto distinto de objetos que se comportan según unas reglas y políticas características del dominio" *[Shlaer, 92]*.

[8] Grupo de objetos interconectados por medio de muchas relaciones, que tienen relativamente poca relación con los objetos de otros grupos *[Shlaer, 92]*.

- *Diagramas transición estado (STD).* Descripción del ciclo de vida de los objetos.
- *Modelo de comunicación de objetos (OCM – Object Communication Model).* Representación de la comunicación de los eventos asíncronos entre los modelos de estado y las entidades externas.
- *Diagramas de thread de control*[9]. Especificación del orden de los eventos y estados asociados con el suceso de un thread de control.
- *Diagramas de flujo de datos acción (ADFD – Action Data Flow Diagram).* Representación de las unidades de procesamiento correspondientes a una acción y de la comunicación entre dichas unidades de procesamiento.
- *Modelos de acceso a los objetos (OAM – Object Access Model).* Visión complementaria del OCM, a modo de compendio de la comunicación síncrona entre los modelos de estado y las instancias de objetos.
- Estas herramientas gráficas se complementan con especificaciones textuales de los objetos, atributos, relaciones, acciones, reglas de transición, eventos, procesos y demás elementos identificados en la aplicación de dichas técnicas.

- **Fase de diseño.**
  - *Diagramas de clases.* Descripción de la vista externa de clases individuales.
  - *Diagramas de estructura de clases.* Detalle de la estructura interna del código correspondiente a las operaciones que realizan las clases.
  - *Diagramas de dependencia.* Representación de las relaciones cliente/servidor y 'de amigos'[10] existentes entre clases.
  - *Diagramas de herencia.* Representación de las relaciones de herencia existentes entre clases.

---

[9] "Un thread de control es la secuencia de acciones y eventos que se producen en respuesta a la llegada de un evento particular no solicitado cuando el sistema está en un estado particular" *[Shlaer, 92]*.

[10] Traducción de friends. Esta relación se emplea cuando un módulo de una clase bien invoca una operación interna de otra clase, o bien realiza un acceso directo a los datos de otra clase.

Metodología

La metodología OL cubre las fases de análisis, diseño e implementación del ciclo de vida de desarrollo del software, según los siguientes pasos:

- **Construcción del modelo de análisis.**
  - *Construcción del modelo de información.* El modelo de información define las unidades conceptuales importantes en el mundo real y para el sistema. Identifica y describe los objetos y sus atributos, define los identificadores que distinguen cada instancia de un objeto, modela, formaliza y compone las relaciones entre objetos, y describe los subtipos y supertipos.
  - *Construcción del modelo de estados.* El modelo de estados describe la secuencia de estados por la que pasan los objetos desde su creación hasta su desaparición. Los ciclos de vida de los objetos con comportamiento dinámico se describen mediante STDs, y adicionalmente, con tablas de transición estado (STT - State Transition Table), que relacionan los estados y eventos, describen todas las acciones, y establecen la lista de eventos. La dinámica del sistema global se presenta en el OCM.
  - *Construcción del modelo de procesos.* El modelo de procesos disecciona las acciones del modelo de estados en procesos fundamentales, descritos estos últimos en un ADFD. Todos los procesos y las acciones que los emplean quedan documentados en la tabla de procesos de estado (STP – State Process Table). Finalmente se representa la comunicación síncrona entre los modelos de estado del sistema en el correspondiente OAM.
  - *Verificación del análisis.* Los modelos de análisis proporcionan un método formal para verificar el comportamiento específico del sistema: simular la ejecución de los modelos.
- **Construcción del modelo de diseño. (Diseño recurrente).**
  - *Definición de los componentes de la arquitectura.* Los elementos y reglas estructurales de la arquitectura se describen con la notación OODLE, la cual utiliza cuatro tipos de diagramas: diagramas de clases, de estructura de

clases, de dependencia y de herencia. Lo ideal es que estos diagramas sean arquetipos, es decir, diagramas que efectúen la especialización mediante una simple sustitución del nombre.

- *Transformación de la aplicación en arquitectura.* Los elementos del dominio de aplicación (atributos, eventos, procesos, etc.) se transforman en instancias específicas de los objetos de la arquitectura. Estas instancias se obtienen especializando los diagramas de clases y de estructura de clases.
- *Transformación de la arquitectura en implementación.* Los módulos del diagrama de estructura de clases (arquetipo) se expresan en plantillas de código en el lenguaje de implementación elegido. Estas plantillas se rellenan con la información de la base de datos de la arquitectura, o a partir de los diagramas de estructura de clases.

Conclusión

OI completa un riguroso análisis antes de pasar al diseño, lo cual reduce el número de fallos y proporciona unas robustas especificaciones. Además, *reduce y controla la iteración en el análisis* al confinarlo a un dominio en cada caso. Dicha reducción y control se hacen extensivos al diseño, ya que *las modificaciones se realizan en el dominio de la arquitectura y se propagan a todo el sistema a través de los arquetipos.*

1.4.2.3 OMT (Object Modeling Technique).

OMT */Rumbaugh, 96/* es una metodología de ingeniería del software que *cubre el ciclo de vida completo del software*, aunque en su bibliografía haga más hincapié en las etapas de concepción, análisis, diseño e implementación.

No obstante, Rumbaugh aclara que describe el proceso de desarrollo utilizando las etapas del ciclo de vida tradicional, pero teniendo muy presente que dicho desarrollo es iterativo y incremental */Rumbaugh, 94/*.

Modelos y técnicas de modelado

OMT propone modelar el sistema desde **tres perspectivas,** aunque relacionadas e imprescindibles todas ellas para realizar una descripción completa del sistema:

- **Modelo de objetos.** Describe la estructura estática del sistema en términos de objetos y relaciones, que se corresponden con las entidades del mundo real. La notación que emplea es una *extensión derivada del modelo entidad relación.*
- **Modelo dinámico.** Presenta los aspectos de control del sistema, que tratan de la temporización y secuencia de operaciones, y de la organización de sucesos y de estados. Gráficamente se representa mediante *diagramas de estado.*
- **Modelo funcional.** Especifica el significado de las operaciones y restricciones del modelo de objetos y las acciones del modelo dinámico, sin tener en cuenta las secuencias, decisiones o estructura de los objetos. Los *diagramas de flujo de datos* resultan especialmente útiles para mostrar las dependencias funcionales.

Metodología

La metodología de desarrollo propuesta por OMT distingue las siguientes **etapas:**

- **Análisis.** El objetivo es desarrollar un *modelo del sistema* expresado mediante objetos y relaciones, flujos dinámicos de control y transformaciones funcionales. Para ello identifica los siguientes pasos:
  - *Definición del problema.* Establecer los requisitos.
  - *Construcción del modelo de objetos.* Identificar las clases y objetos relevantes del dominio, preparar un diccionario de datos, identificar las asociaciones entre clases, describir los atributos de objetos y asociaciones, simplificar las clases aplicando herencia, y verificar los caminos de acceso. El modelo se itera y refina, y se agrupan las clases en módulos que soportan un subconjunto lógico del modelo.
  - *Construcción del modelo dinámico.* Preparar los escenarios, identificar los eventos, preparar la traza de eventos para cada escenario, construir un

diagrama de estados para cada clase, y equiparar los eventos entre objetos para verificar su consistencia y completitud al nivel de sistema.

- *Construcción del modelo funcional.* Identificar los valores de entrada y salida, construir los DFDs, describir las funciones, identificar las restricciones funcionales, y especificar los criterios de optimización.
- *Iteración del análisis.* Documentar, verificar, iterar y refinar los tres modelos obtenidos.

- **Diseño del sistema.** Durante esta etapa se construye una *arquitectura para el sistema.* Los pasos para obtenerla son los siguientes:
  - *Descomposición del sistema en subsistemas.* Identificar aquellos paquetes de clases, asociaciones, operaciones, eventos y restricciones que estén relacionados y posean una interface bien definido con otros subsistemas.
  - *Identificación de la concurrencia.* Determinar cuales son los objetos agrupables en un mismo thread de control.
  - *Asignación de subsistemas a procesadores y tareas.* Estimar las necesidades de rendimiento, elegir la implementación, asignar subsistemas software a procesadores, y determinar la conectividad de las unidades físicas.
  - *Gestión de los almacenamientos de datos.* La elección del almacenamiento debe considerar el coste, el tiempo de acceso, la capacidad y la fiabilidad.
  - *Identificación de los recursos globales.* Definir los recursos y determinar los mecanismos que permiten controlar su acceso.
  - *Elección del mecanismo de control del software.* Elegir el tipo de flujo de control (externo o interno).
  - *Manejo de situaciones límite.* Considerar las situaciones de inicialización, terminación y fallo para garantizar la manipulación correcta de los recursos.
  - *Establecimiento de prioridades.* Elegir entre los objetivos que se consideran deseables pero son incompatibles.

- **Diseño de objetos.** El énfasis en el dominio de aplicación se desplaza hacia los *conceptos informáticos,* según los siguientes pasos:

- *Combinación de los modelos.* Convertir las acciones y actividades del modelo dinámico y los procesos del modelo funcional en operaciones vinculadas a las clases del modelo de objetos.
- *Diseño de los algoritmos.* Elegir los algoritmos, seleccionar las estructuras de datos, definir las clases internas, y asignar responsabilidades para las operaciones a las clases apropiadas.
- *Optimización del diseño.* Optimizar el modelo de análisis para que la implementación sea más eficiente.
- *Implementación del control.* Refinar la estrategia para implementar los modelos basados en eventos y estados presentes en el modelo dinámico.
- *Ajuste de la herencia.* Aumentar la herencia mediante el ajuste de las definiciones de clases y operaciones.
- *Diseño de las asociaciones.* Formular una estrategia para implementar las asociaciones del modelo de objetos.
- *Representación de los objetos.* Decidir cuando emplear tipos primitivos o combinar grupos de objetos relacionados.
- *Empaquetado físico.* Agrupar clases y asociaciones en módulos.

- **Implementación.** Las clases de objetos y las relaciones desarrolladas durante el diseño se traducen a un lenguaje de programación concreto.

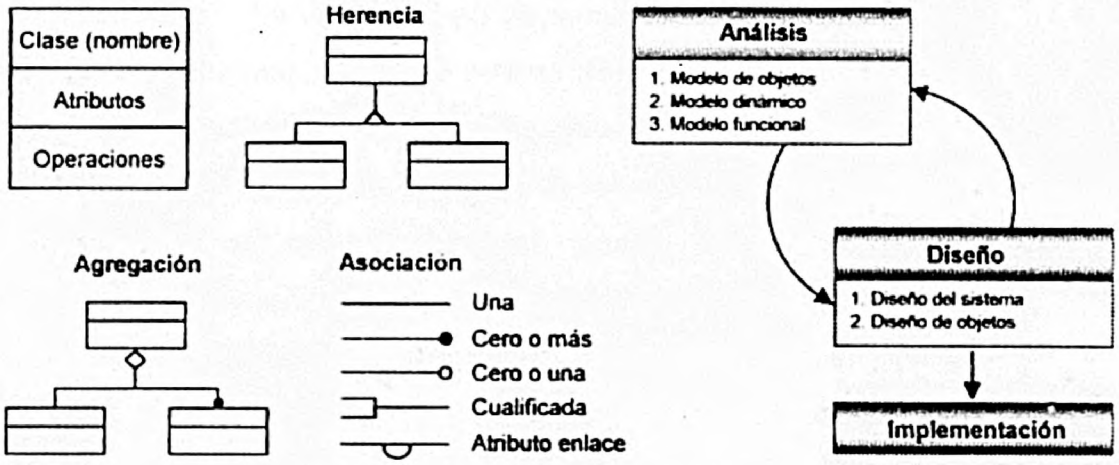


Figura 1.18: Notación básica y metodología OMT.

Conclusión

OMT propone un *proceso flexible.* Permite *iterar entre el análisis y diseño* (figura 1.18) para acelerar la construcción del software, y facilita la *realización de prototipos* de las funcionalidades claves. Quizás estos sean los motivos que la han consolidado como una *metodología líder en la industria.*

1.4.2.4 OOSE (Object-Oriented Software Engineering).

OOSE *[Jacobson, 92]*, versión simplificada de Objectory *[Objectory, 96], combina la programación orientada a objetos, el modelado conceptual y el diseño en bloques.*

Modelos y técnicas de modelado

Jacobson et al. consideran que la gestión de la complejidad del sistema se puede realizar *introduciendo dicha complejidad de forma gradual,* según un orden específico, en los sucesivos modelos que proponen:

- **Modelo de requisitos.** Delimita el sistema y define su funcionalidad. Consta de los siguientes elementos:
  - *Modelo de casos de uso.* Modelo central del cual se derivan los restantes. Describe los actores y los "casos de uso" (traducción de "use cases"). Los actores establecen el papel del usuario en el intercambio de información, mientras que los casos de uso especifican la funcionalidad del sistema.
  - *Modelo de objetos del dominio del problema.* Perspectiva lógica del sistema que contempla los objetos propios del dominio del problema, proporcionando la lista de nombres para especificar los casos de uso.
  - *Descripciones de las interfaces.* Define detalladamente las interfaces con el usuario y con otros sistemas.
- **Modelo de análisis.** Estructura el modelo de requisitos mediante el modelado de tres tipos de objetos: objetos interface, objetos entidad y objetos de control.

- **Modelo de diseño.** Adapta el modelo de análisis al entorno de implementación. El modelo de diseño consta de:
  - *Diagrama de interacciones.* Describe cada caso de uso concreto en términos de objetos que se comunican. La comunicación se modela como bloques que se envían estímulos.
  - *Gráficos de transición estado.* Representan el comportamiento del objeto indicando los estímulos que puede recibir y enviar.
- **Modelo de implementación.** Consiste en el código fuente de los objetos especificados en el modelo de diseño.
- **Modelo de pruebas.** Resultado de realizar las pruebas al modelo de implementación. Los conceptos principales son: la especificación de las pruebas y los resultados de la ejecución de dichas pruebas.

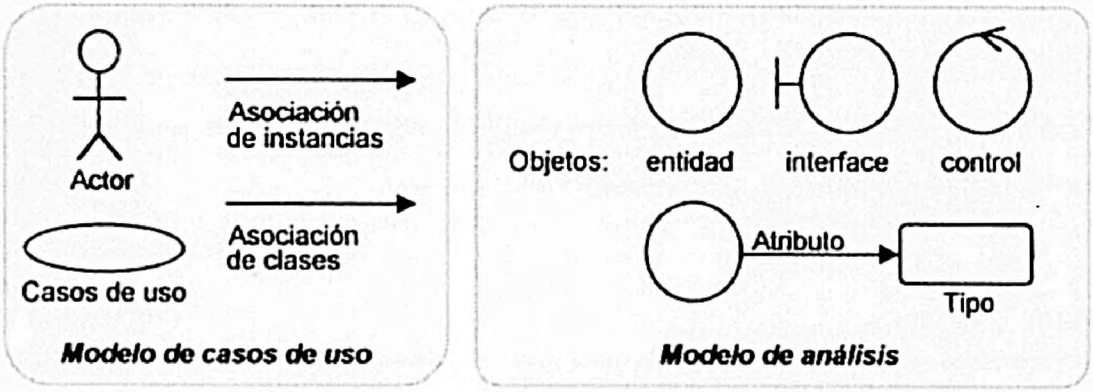


Figura 1.19: Notación básica OOSE.

Metodología

El proceso de desarrollo que propone OOSE es iterativo e incremental, pudiendo llegar a solaparse en algunos casos las distintas **etapas** que contempla:

- **Análisis.** OOSE no describe los pasos necesarios para obtener los dos modelos resultantes del análisis: el modelo de requisitos y el modelo de análisis.
- **Construcción.** Esta etapa consta de dos pasos:

- *Diseño.* Los objetos de análisis se traducen a objetos de diseño considerando el entorno de implementación, y se establecen las interfaces de los objetos de acuerdo con las interacciones presentes en los distintos casos de uso.
- *Implementación.* Cada objeto se codifica en el lenguaje de programación específico.

- **Pruebas.** Tras realizar la planificación, se describen y se especifican las pruebas. Los resultados de la ejecución de las pruebas se recogen en una tabla de decisión, y en el caso de resultados fallidos, se realizan las modificaciones necesarias. El orden de ejecución de las pruebas es el siguiente: módulos objeto y bloques (bajo nivel), casos de uso, y sistema.

Conclusión

La principal contribución de Objectory, y por ende de OOSE, es la introducción del concepto de casos de uso. La visión del sistema mediante casos de uso *es un potente método para analizar el sistema* en términos de las demandas que se le solicitan y el procesamiento necesario para dar respuesta a las mismas.

1.4.2.5 Fusion.

Fusion *[Coleman, 94] integra y amplia las mejores características de varios enfoques orientados a objeto y de métodos formales,* para proporcionar un acceso más directo del análisis al diseño y a la implementación.

Modelos y técnicas de modelado

Las técnicas de modelado utilizadas en Fusion son varias: el análisis comienza con los modelos de objetos y finaliza con los modelos de operación; el diseño comienza con los gráficos de interacciones entre objetos y finaliza con los gráficos de herencia. Todas estas técnicas se resumen a continuación:

- **Modelos de objetos.** Captura la estructura estática (conceptos y relaciones) del dominio del problema. Su notación es similar a la de los diagramas E/R.

- **Modelos de objetos del sistema.** Subconjunto del modelo de objetos que atañe al sistema, excluyendo todas las clases y relaciones que pertenecen al entorno.
- **Modelos de interface.** Define la comunicación del sistema en términos de eventos y cambios de estado resultantes, distinguiendo los siguientes modelos:
  - *Modelos de ciclo de vida.* Caracteriza la secuencia de interacciones (operaciones y eventos) en la que participa el sistema en su tiempo de vida.
  - *Modelos de operación.* Describe los efectos de cada operación del sistema, indicando los cambios de estado y los eventos de salida que genera.
- **Gráficos de interacciones entre objetos.** Existe un gráfico por operación, que define la secuencia de mensajes que se intercambian los objetos para realizarla.
- **Gráficos de visibilidad.** Representan la estructura de referencias de las clases, identificando para cada una de estas clases: los objetos que las instancias de la clase necesitan, y los tipos de referencias apropiados para estos objetos.
- **Descripción de clases.** Cada clase tiene su descripción asociada, incluyendo sus métodos, atributos y valores de estos atributos.
- **Gráficos de herencia.** Refleja las relaciones de herencia entre las clases.

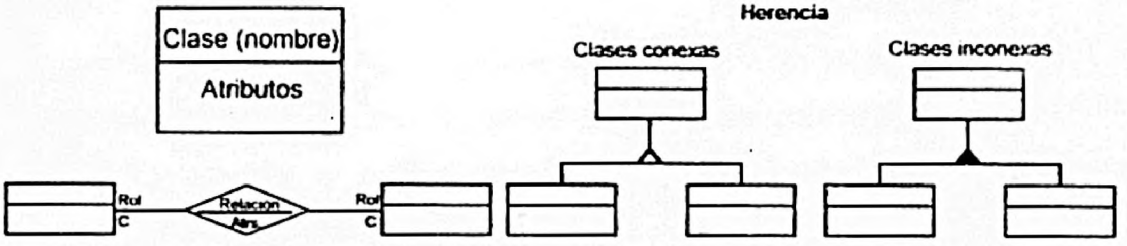


Figura 1.20: Notación básica del modelo de objetos de Fusion.

Metodología

La metodología (figura 1.21) propuesta por Fusion se estructura en **tres etapas:**

- **Análisis.** El énfasis recae en la descripción del dominio y de su comportamiento externo, de acuerdo con los siguientes pasos:

- *Desarrollo del modelo de objetos.* Tras la obtención de la lista de clases y relaciones candidatas, se incorpora esta información al diccionario de datos y se construye el modelo de objetos de forma incremental.
- *Determinar la interface del sistema.* La información sobre los agentes, las operaciones y los eventos identificados se añade al diccionario de datos, y se construye el modelo de objetos del sistema.
- *Desarrollo del modelo de interface.* Los escenarios se generalizan y se elaboran los modelos de ciclo de vida y de operación.
- *Comprobación de los modelos de análisis.* Los modelos de análisis deben ser completos y consistentes.

- **Diseño.** Las definiciones abstractas obtenidas en el análisis se transforman en estructuras software, según los siguientes pasos:
  - *Construcción de los gráficos de interacciones entre objetos.* Identificados los objetos más relevantes, se distinguen controladores y colaboradores, se establecen los mensajes de intercambio, y se registra el modo de interacción.
  - *Construcción de los gráficos de visibilidad.* Partiendo de los gráficos anteriores, se identifica el tipo de referencia de visibilidad, y se desarrolla el gráfico de visibilidad.
  - *Descripción de las clases.* Los métodos y parámetros se extraen de los gráficos de interacciones entre objetos, los atributos se obtienen del modelo de objetos del sistema y del diccionario de datos, los atributos de los objetos se derivan de los gráficos de visibilidad, y la información sobre herencia se recoge de los gráficos de herencia.
  - *Construcción de los gráficos de herencia.* Las generalizaciones y especializaciones se estudian en el modelo de objetos, los métodos comunes se localizan en los gráficos de interacciones y las descripciones de las clases, y las referencias de visibilidad se obtienen de los gráficos de visibilidad.
  - *Actualización de las descripciones de las clases.* Con la información de los gráficos de herencia se actualizan las descripciones de las clases.

- **Implementación.** Los pasos que se contemplan en esta etapa son los siguientes:
  - *Codificación.* Los ciclos de vida se traducen en máquinas de estados, se implementan dichas máquinas de estados, y se codifican las clases con sus atributos, métodos y relaciones.
  - *Ejecución.* Un buen rendimiento del sistema es un factor de suma importancia, que frecuentemente fuerza a la optimización del código usado.
  - *Revisión.* Las pruebas e inspecciones tienen como objeto la detección de los defectos del software.

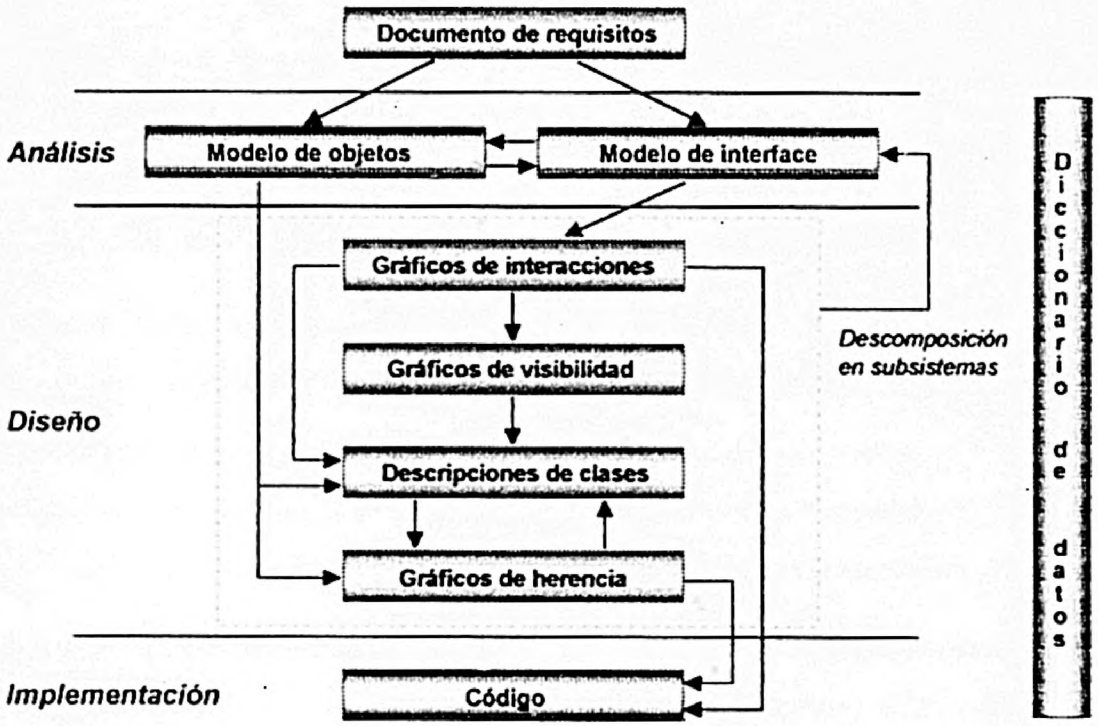


Figura 1.21: Metodología Fusion.

Conclusión

Fusion propone un proceso sistemático, que establece los puntos de toma de decisión y los entregables de cada etapa. Es tan riguroso que *incluso permite detectar las inconsistencias entre los modelos y proporciona las reglas para comprobar que todos los modelos construidos son consistentes y completos.*

1.4.2.6 IDEF4.

IDEF4 *[IDEF4, 95]* es un *método de diseño para el desarrollo de sistemas cliente/servidor basados en componentes*, que establece tres niveles de diseño para reducir la complejidad del sistema: nivel de sistema, nivel de aplicación y bajo nivel.

Modelos y técnicas de modelado

Toda la información de modelado del sistema se captura mediante **tres modelos y un fundamento de diseño** que se presentan a continuación:

- **Modelo estático.** Especifica la estructura estática de los objetos de diseño, es decir, los objetos, sus atributos, y las relaciones entre los mismos. Consta de:
  - *Diagramas de estructuras.* Contienen los objetos relacionados por herencia, enlace y relación. Se pueden especializar para que incluyan una relación.
- **Modelo dinámico.** Describe las relaciones dinámicas entre objetos mediante eventos y paso de mensajes. Este modelo engloba dos tipos de diagramas:
  - *Diagramas cliente servidor.* Ilustra las relaciones de uso entre objetos.
  - *Diagramas de estado.* Representa el comportamiento de un objeto, indicando sus estados y las transiciones generadas por ciertos eventos.
- **Modelo de comportamiento.** Presenta la implementación de mensajes mediante métodos en los objetos, para lo cual se emplean:
  - *Diagramas de comportamiento.* Describen las relaciones entre los métodos que implementan un mensaje, y permiten discernir las similitudes en comportamientos para reutilizar las especificaciones de dichos métodos.
- **Fundamento de diseño.** Proporciona una vista top-down del sistema y documenta los cambios de diseño. También describe las transformaciones de hitos en productos de diseño mediante diagramas. Cada situación de diseño se representa por una caja redondeada etiquetada en la parte superior y una lista de diagramas que definen la situación del modelo.

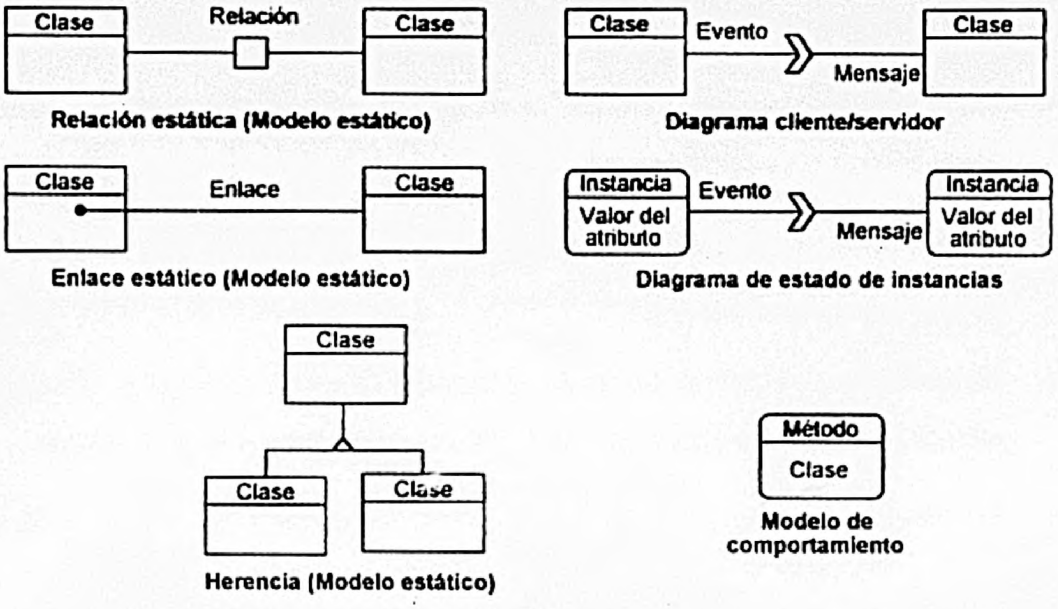


Figura 1.22: Notación básica de IDEF4.

Metodología

Su proceso de desarrollo (figura 1.23) distingue las siguientes etapas:

- **Análisis de requisitos.** El desarrollo de los modelos de dominio y de requisitos del sistema se realiza basándose en objetos, relaciones y escenarios (también denominados casos de uso). Los escenarios son muy importantes para establecer los requisitos funcionales, validar el diseño y verificar la implementación.

- **Diseño del sistema.** Conocidos los objetos, se determina el contexto, se asegura la conectividad con los sistemas existentes y se identifican las aplicaciones para satisfacer los requisitos. También se construyen los modelos estático, dinámico, de comportamiento y de fundamento de diseño al nivel de sistema.

- **Diseño de la aplicación.** En esta etapa se identifican y especifican todos los componentes software (particiones), y se construyen los modelos estático, dinámico, de comportamiento y de fundamento de diseño al nivel de aplicación.

- **Diseño a bajo nivel.** Identificados los objetos de bajo nivel, se construyen los correspondientes modelos estático, dinámico, de comportamiento y de fundamento de diseño.

En cada etapa, IDEF4 prescribe un **procedimiento iterativo** que consta de cinco pasos:

- **Partición.** División del diseño en piezas más pequeñas y manejables utilizando como criterio el comportamiento.
- **Clasificación/especificación.** Los objetos y sus comportamientos se comparan con modelos existentes para clasificarlos.
- **Ensamblaje.** Los objetos de diseño se incorporan en las estructuras existentes, identificando las relaciones, enlaces e interacciones dinámicas entre objetos.
- **Simulación.** Los modelos se validan y verifican mediante la ejecución de los casos de uso descritos en el modelo de requisitos.
- **Reorganización.** Los nuevos objetos introducidos obligan a reorganizar las estructuras existentes. Este ajuste del diseño origina una nueva iteración.

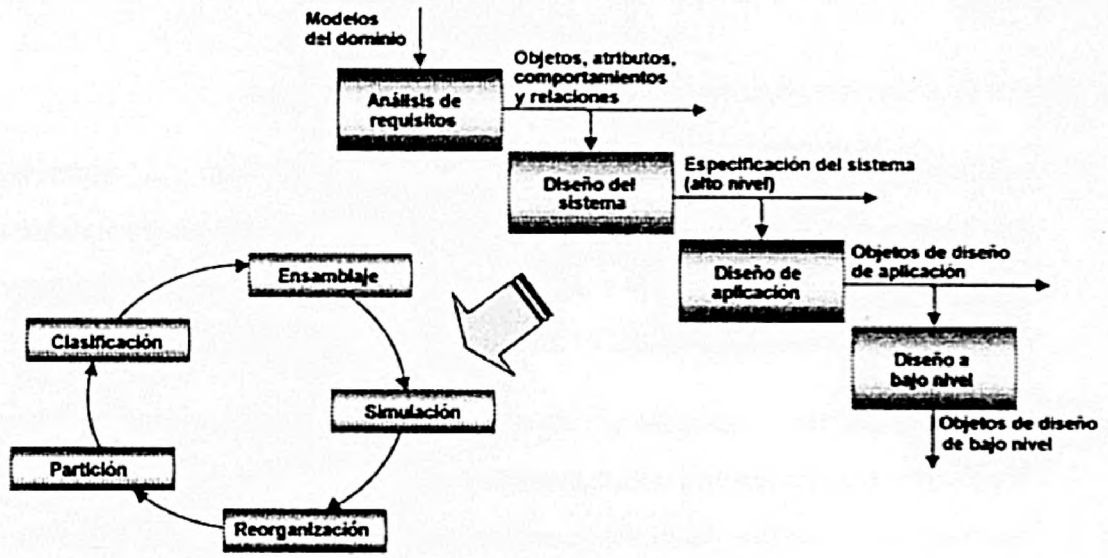


Figura 1.23: Metodologia IDEF4.

Conclusión

IDEF4 propone un proceso de desarrollo que permite una *transición suave desde los modelos de análisis de requisitos y de dominio de la aplicación hasta el diseño y generación de código fuente.* Además, *el fundamento de diseño recoge el modo en que se realiza este proceso, justificando todas las decisiones tomadas en el mismo.*

1.4.2.7 UML (Unified Modeling Language).

UML *[UML, 97]* es un *lenguaje para la especificación, visualización, construcción y documentación de sistemas software, así como para el modelado de organizaciones y de sistemas no software.* Representa la unificación de los métodos OOADA, OMT y Objectory, añadiendo ideas de otros muchos métodos *[Booch, 97]*.

Técnicas de modelado

UML contempla los siguientes diagramas: diagramas de casos de uso, de clases, de transición estado, de interacciones, de componentes, y de distribución. Todos estos diagramas son adaptaciones de los ya comentados en los métodos predecesores.

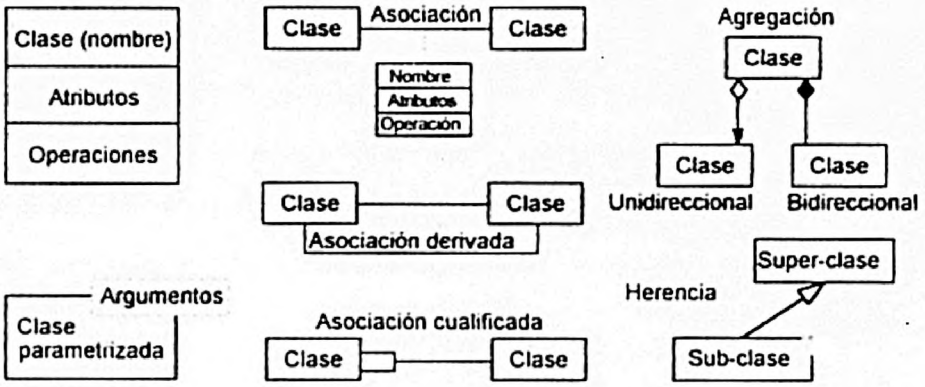


Figura 1.24: Notación básica del diagrama de clases UML.

No estandariza un proceso de desarrollo

Los autores pretenden preservar este lenguaje independiente de cualquier proceso, pero aconsejan un proceso iterativo e incremental, guiado por casos de uso, y centrado en una arquitectura genérica, como Rational Objectory *[Rational, 97]*.

Conclusión

La especificación UML *completa y mejora las técnicas de modelado de los métodos que unifica*, y ha sido *adoptada oficialmente por la OMG* (Object Management Group) *[OMG, 97]*, cuyo objetivo es promocionar la tecnología orientada a objetos.

1.4.3 Análisis de los métodos orientados a objetos desde la perspectiva de la reutilización.

Todas las metodologías orientadas a objeto descritas en los apartados anteriores (OOADA, OL, OMT, OOSE, Fusion e IDEF4) propugnan la reutilización como el principal beneficio, pero sus autores son conscientes de que la reutilización no sucede de forma automática, ni aún usando tecnología orientada a objetos. Pese a ello, todas estas metodologías adolecen de procesos específicos que soporten la reutilización más allá de la **reutilización de componentes**, aunque sus construcciones básicas y notaciones gráficas sean adecuadas en la mayoría de ellas.

En las tablas 1.3, 1.4 y 1.5 se comparan las distintas metodologías orientadas a objetos desde los siguientes puntos de vista:

- **Construcciones básicas y sus notaciones gráficas.** En la tabla 1.3 se indican los conceptos que contribuyen a la reutilización y que soportan cada metodología, y en la tabla 1.4 si una metodología es capaz de documentar estas construcciones o conceptos. Se puede observar que la mayoría de estas metodologías:
  - *Soportan las construcciones básicas señaladas, salvo OL quizás por su gran incidencia en el modelo dinámico.*
  - *Disponen de notaciones gráficas satisfactorias, aunque éstas difieran bastante de una metodología a otra, debido posiblemente al número de construcciones básicas y al conjunto de notaciones empleadas por cada construcción básica.*
- **Procesos.** Se comparan atendiendo a la integración de la reutilización en los procesos de análisis y diseño. Una buena metodología debe suministrar las guías para construir y utilizar componentes reutilizables a todos los niveles. A este respecto, se constata (tabla 1.5) que:
  - *La mayoría de las metodologías estudiadas soportan la reutilización de componentes, pero se olvidan de los aspectos de gestión de los mismos.*

- *Fusion e IDEF4 soportan el desarrollo con/para reutilizar componentes con mayor nivel de abstracción, si bien IDEF4 obvia la gestión de los mismos*

Para justificar el interés de una metodología con reutilización sistemática, se analiza detalladamente la manera en que las metodologías estudiadas aplican la reutilización y los esfuerzos realizados por el consorcio SER (Software Evolution and Reuse) para promover la integración de la reutilización en todas y cada una de las etapas del ciclo de desarrollo de un proyecto software.

Construcciones básicas	*OOADA*	*LO*	*OMT*	*OOSE*	*Fusion*	*IDEF4*
Clases/objetos	✓	✓	✓	✓	✓	✓
Meta-clases	✓		✓	✓	✓	✓
Clases genéricas[11]	✓					
Clases abstractas[12]	✓		✓	✓	✓	✓
Encapsulación	✓	✓	✓	✓	✓	✓
Herencia simple	✓	✓	✓	✓	✓	✓
Herencia múltiple	✓	✓	✓	✓	✓	✓
Asociación	✓	✓	✓	✓	✓	✓
Agregación	✓	I[13]	✓	✓	✓	✓
Métodos/mensajes	✓		✓	✓	✓	✓
Módulos	✓	✓	✓	✓	I	✓
Subsistemas	✓	✓	✓	✓	✓	✓
Frameworks[14]	✓		I	✓	✓	✓

Tabla 1.3: Comparación de los conceptos de las metodologías orientadas a objetos.

---

[11] Una clase genérica es una "plantilla" para crear clases especificas sin más que proporcionar los parámetros necesarios.

[12] Una clase abstracta es aquella cuya finalidad no es crear instancias

[13] "I" significa implicito en la discusión de la metodología.

[14] Un framework es una colección de clases que proporciona un conjunto de servicios para un dominio.

Notación gráfica	*OOADA*	*OL*	*OMT*	*OOSE*	*Fusion*	*IDEF4*	*UML*
Clases/objetos	✓	✓	✓	✓	✓	✓	✓
Meta-clases	✓		✓	✓	✓	✓	✓
Clases genéricas	✓						✓
Clases abstractas	✓		✓	✓	✓	✓	✓
Encapsulación					✓	✓	✓
Herencia simple	✓	✓	✓	✓	✓	✓	✓
Herencia múltiple	✓	✓	✓	✓	✓	✓	✓
Asociación	✓	✓	✓	✓	✓	✓	✓
Agregación	✓		✓	✓	✓	✓	✓
Métodos/mensajes	✓		✓	✓	✓	✓	✓
Módulos	✓					✓	✓
Subsistemas	✓			✓	✓	✓	✓
Frameworks	✓			✓	✓	✓	✓

Tabla 1.4: Comparación de las notaciones de las metodologías orientadas a objetos.

Procesos	*OOADA*	*OL*	*OMT*	*OOSE*	*Fusion*	*IDEF4*
Considera la reutilización	✓	✓	✓	✓	✓	✓
Desarrollo con reutilización:						
componentes	I	I	I	✓	✓	✓
otros	I	I			✓	✓
Desarrollo para reutilizar:						
componentes	✓	I	I	✓	✓	✓
otros	✓	I			✓	✓
Gestión de:						
componentes	I			✓	✓	
otros					✓	

Tabla 1.5: Comparación de los procesos de las metodologías orientadas a objeto.

Análisis de OOADA

Esta metodología incluye una actividad explícita de **identificación de patrones comunes** (diseño para reutilizar) en el micro proceso. Estos patrones pueden hacer referencia a clases, objetos, diseños o frameworks.

En términos *de diseño con reutilización, no hace mención explícita.* Sin embargo, destaca las ventajas y presenta ejemplos pero sin dar recomendaciones. También *aboga por la utilización de generadores de aplicación.*

Los aspectos de gestión de componentes reutilizables tampoco son considerados explícitamente, aunque presenta como ejemplo la construcción de una librería de clases básicas, que permite extraer e incluso, generalizar numerosas ideas.

Análisis de OL

OL contempla la **reutilización de procesos** sobre la base de *dos aspectos de análisis:*

- *La propia definición de proceso.* Un proceso incorpora tanto la transformación de datos como cualquier actividad para leer o escribir datos de los almacenamientos. Se asume que un proceso no requiere una tarea especial para acceder a los datos, ya que se obtiene mediante un evento u otro proceso.
- *Los pasos de análisis.* Primero se construyen los modelos de estado y luego, los modelos de procesos (ADFDs) para las actividades identificadas en los modelos de estados. Debido a este proceder, es muy común encontrar el mismo proceso usado en varios ADFDs de un único o de varios modelos de estados.

Estas consideraciones sobre procesamiento y reutilización aseguran *que si un proceso es reutilizable en los ADFDs, puede ser transformado en código reutilizable.*

Por otra parte, su estrategia recursiva de diseño establece la división inicial del sistema en dominios e incluso, subsistemas. Cada división se desarrolla por separado hasta los pasos finales de construcción. Esto significa que puede trasladarse intacto a otro sistema, obteniendo así la **reutilización del dominio completo.**

Análisis de OMT

La metodología OMT distingue *dos tipos de reutilización:*

- *Compartición de código recién escrito dentro de un proyecto.* Simplemente precisa descubrir las secuencias de código redundantes en el diseño, y utilizar las capacidades del lenguaje de programación para compartir su implementación.
- *Reutilización de código previo en proyectos nuevos.* En este caso, la reutilización de clases aisladas es improbable, siendo lo normal la reutilización de subsistemas cuidadosamente pensados.

También sugiere unas **reglas de estilo para la reutilización** centradas principalmente en los métodos. Además, reitera las prácticas de la programación estructurada, enfatizando estilos como la encapsulación con mínimo acoplamiento y máxima cohesión.

Análisis de OOSE

Los modelos generados proporcionan una buena base para *reutilizar información:* el modelo de casos de uso se reutiliza para construir todos los modelos subsecuentes, y el modelo de análisis persigue la creación de una estructura robusta de objetos reutilizables ante cambios futuros.

No obstante, se centra en la **reutilización de componentes** software. Los componentes son un mecanismo de refuerzo para el lenguaje de implementación, ya que permiten el desarrollo de las aplicaciones a partir de construcciones con un mayor nivel de abstracción que las primitivas. Esto también significa que *el desarrollador debe tener tan buen conocimiento de la librería de componentes como del propio lenguaje.*

En relación con la **gestión de componentes,** distingue dos tipos de actividades:

- *Construcción de componentes individuales.* Sugiere algunos criterios para diseñar componentes reutilizables, recuerda criterios propuestos por otros autores, y hace especial énfasis en la documentación de estos componentes.

- *Diseño de librerías de componentes.* La construcción y evolución de una librería involucra tareas de selección, clasificación y gestión de los componentes existentes, sin olvidar la difusión de información actualizada de la misma.

Análisis de Fusion

En lo que respecta al **desarrollo de componentes para reutilizar,** *se analizan las fases de desarrollo e identifican los distintos tipos de componentes reutilizables:*

- *Análisis.* Precisa actividades explícitas para la construcción de modelos de dominios en lugar de modelos de sistemas específicos.
- *Diseño.* Identifica dos tipos de componentes reutilizables: las clases abstractas y los frameworks.
- *Implementación.* La mayoría de los frameworks y/o grupos de clases se transforman en librerías de código orientado a objetos.

A la hora de **desarrollar con reutilización,** Fusion distingue *dos posibilidades: la configuración de un framework, y la integración de subsistemas.* En ambos casos es imprescindible disponer de infraestructura de reutilización, entendiendo como tal: estándares para documentar y asegurar la calidad de los componentes, y herramientas de soporte para recuperar y entender los componentes. Ninguna de estas dos facetas está bien soportada en los entornos actuales de producción de software.

Análisis de IDEF4

En el propio desarrollo de IDEF4 han prestado mucha atención a aspectos relacionales con la reutilización, concretamente: **reutilización de los sistemas existentes** por medio de la encapsulación, **creación de diseños y componentes software reutilizables, e inclusión de componentes comerciales reutilizables** (COTS - Commercial-Off-The-Shelf) en los diseños.

El nivel de detalle de los objetos de diseño permite la *obtención inmediata del código fuente. Cada componente de diseño tiene una especificación asociada que engloba:*

- *La especificación interna.* Definición de la implementación interna del componente de diseño.
- *La especificación externa.* Caja negra que define el comportamiento externo, los protocolos y las responsabilidades del componente de diseño. También permite encapsular el software heredado o los componentes comerciales.

Conclusión. Esfuerzos del consorcio SER

Por lo tanto, ninguna de las metodologías analizadas integra la reutilización en todas y cada una de las etapas del ciclo de desarrollo de un proyecto software.

El consorcio SER pretende resolver esta situación a través de una serie de proyectos ESPRIT *[Paci, 96]:* REBOOT (reutilización basada en técnicas orientadas a objeto), PROTEUS (soporte para la evolución de sistemas), RECYCLE (entorno integrado para soportar las actividades de mantenimiento), EUROWARE (reutilización en áreas más extensas), EUROBANQUET (soporte para una evolución efectiva del software en los bancos europeos), COSMOS (análisis de la estructura, complejidad y calidad del producto software) y SCALE (soporte a la reutilización de componentes a gran escala).

Mención especial merece el proyecto **REBOOT** (REuse Based on Object-Oriented Techniques) *[Karlsson, 95]* ya que aborda los aspectos organizativos y metodológicos para el éxito de la reutilización del software:

- Asesora en la forma de construir una estrategia para introducir la reutilización.
- Proporciona un Modelo de Madurez de la Reutilización (RMM – Reuse Maturity Model) que permite valorar la introducción y mejora de la reutilización.
- Propone guías para las etapas del ciclo de desarrollo para/con reutilización.
- Incluye tareas de reingeniería de componentes existentes, su cualificación y clasificación para su posterior uso.
- Asesora sobre la forma de gestionar proyectos con reutilización, de gestionar los elementos de un repositorio y de medir el efecto de la reutilización (métricas).

1.4.4 Aplicación de la tecnología orientada a objetos al desarrollo del software de control de FMS.

A continuación se presentan algunas metodologías para el desarrollo de software de control de FMSs que emplean la tecnología orientada a objetos. Algunas fomentan la reutilización de componentes, otras de patrones, etc., pero **ninguna de ellas incluye la reutilización en el propio proceso de desarrollo.** También se incluyen aplicaciones de esta tecnología en otros aspectos relacionados con los sistemas de control.

1. Método propuesto por Wu

Wu *[Wu, 94]* considera que los métodos orientados a objeto existentes no son adecuados para el entorno de fabricación, y propone un *método que combina características de los enfoques orientados a objetos puros y de los enfoques estructurados.* Su metodología, de naturaleza iterativa, establece las siguientes fases:

- **Descomposición del sistema.** El sistema se descompone en una jerarquía de áreas funcionales (FS – Function Subjects) que se representan en un diagrama de bloques funcionales (FBD – Function Block Diagram).
- **Identificación de las clases y los objetos.** En cada FS se realiza una identificación más exhaustiva de clases y objetos mediante un diagrama de relación entre subsistemas (SRD – Subsystem Relationship Diagram), y se clasifican las clases y objetos identificados en dos categorías:
  - *Clientes.* Aquellos que requieren servicios de uno o más servidores.
  - *Servidores.* Aquellos que realizan los servicios en ciertas clases y objetos.
- **Especificación de las estructuras estáticas.** Relaciones de herencia y agregación, que describen las interacciones y relaciones entre los objetos.
- **Especificación de las interacciones dinámicas.** El comportamiento dinámico de los objetos se describe mediante: diagramas transición estado (STC – State Transition Chart) que especifican el ciclo de vida, y diagramas de ciclo de actividad (ACD – Activity Cycle Diagram) que formalizan las interacciones.

- **Especificación de las conexiones entre objetos.** Una visión completa del sistema debe recoger otras dos relaciones:
  - *Conexiones de instancias.* La revisión de los STCs y ACDs facilita la tarea de identificación de estas relaciones asociadas con los atributos.
  - *Conexiones de mensajes.* Un esbozo de estas relaciones reside en los SRDs.
- **Integración de los subsistemas.** Los subsistemas se integran en un modelo gráfico completo que combina la información necesaria sobre el dominio.

Para ilustrar su método, Wu presenta un sistema de fabricación bajo pedido (OHMS – Order Handling Manufacturing System); el mismo ejemplo que modela con IDEF0.

Este método emplea las construcciones básicas y la notación gráfica de los métodos existentes. **No establece procesos específicos de desarrollo para/con reutilización, ni trata aspectos de gestión.** El propio autor indica que la metodología está en las primeras etapas de desarrollo, y que *una de las posibles mejoras es la inclusión de la reutilización dentro del proceso de desarrollo.*

2. Método propuesto por Adiga

Adiga *[Adiga, 93]* presenta un enfoque híbrido que *aborda el análisis y diseño de los sistemas empleando descomposición funcional, técnicas de modelado de datos y análisis de los eventos relevantes* para desarrollar las especificaciones de los objetos software. Las **fases** que establece para construir un prototipo son las siguientes:

- **Partición del sistema en subsistemas.** Los requisitos del sistema se obtienen en forma de un conjunto de módulos funcionales o procesos, y la comunicación entre procesos se captura por medio de diagramas de flujo de mensajes (MFDs – Message Flow Diagrams).
- **Identificación de la librería de objetos.** La abstracción de los requisitos comunes a todos los procesos proporciona la especificación para el conjunto de librerías necesarias, incluyendo el nivel de detalle que deben tener los objetos software que compondrán dichas librerías.

- **Descomposición de la librería software.** En esta etapa se diseñan y desarrollan los componentes reutilizables específicos, según los siguientes pasos:
  - *Definición de sus objetivos basándose en los requisitos de las librerías.*
  - *Estudio del entorno de aplicación para obtener datos de modelado.*
  - *Identificación de las entidades, relaciones y atributos, y construcción de un diagrama entidad relación (E/R) normalizado.*
  - *Abstracción de los objetos e instanciación de las variables a partir del diagrama E/R normalizado.*
  - *Identificación de los eventos principales y los objetos afectados por éstos.*
  - *Construcción de un MFD por cada evento para identificar el comportamiento de los objetos participantes y descubrir sus métodos.*
  - *Revisión de las definiciones de los objetos y consolidación de sus especificaciones.* Volver al segundo paso si el modelo no es satisfactorio.
  - *Implementación del diseño.*

Para ilustrar su método de diseño, Adiga lo aplica a un sistema simplificado de fabricación de obleas de semiconductores. Además, presenta **tres aplicaciones industriales** realizadas por dos firmas pioneras en la aplicación de la tecnología orientada a objetos a esta área:

- Consilium Inc. aplica la orientación a objetos en el desarrollo de un **sistema de gestión de planta denominado FlowStream.**
- Savoir[15] aplica la ***metodología orientada a objetos denominada Actors***, mediante la herramienta FLEXIS ToolSet, en dos aplicaciones típicas de **control en tiempo real para una célula de ensamblaje y una línea de cristal.** Actors construye aplicaciones Grafcet (lenguaje gráfico para expresar el flujo de control), que consideran los sistemas de fabricación como un conjunto de actores independientes que realizan las operaciones para fabricar los pedidos.

---

[15] Posteriormente denominada Pacific Software Solutions.

Tanto la metodología propuesta por Adiga como las aplicaciones industriales de las empresas Consilium Inc, y Savoir, emplean la tecnología orientada a objetos para incrementar la reutilización. No aportan construcciones básicas o notaciones gráficas nuevas, ya que usan las disponibles en los métodos existentes.

Todas ellas persiguen la reutilización de componentes pero sin establecer procesos concretos de desarrollo para/con reutilización. Adiga también incluye la reutilización de diseños mediante una arquitectura de control que acompaña a su metodología.

En cuanto a los aspectos de gestión, **todas emplean librerías de componentes pero ninguna propone guías para realizar las tareas de gestión de componentes u otros elementos reutilizables.**

3. Método propuesto por Elia

Elia et al. *[Joshi, 94]* acometen la *construcción del software de control de FMSs mediante un conjunto de patrones.* Cada patrón individual describe un aspecto de diseño específico del problema, aunque en conjunto, el lenguaje de patrones constituya un *método de diseño que aborda el proceso de desarrollo desde el análisis hasta la implementación.*

Cada patrón consta de un conjunto de objetos cooperantes unidos mediante ciertas relaciones fijas. Los patrones identificados son los siguientes: jerarquía de niveles de control, conocimiento y comunicación entre módulos de control, concurrencia en los módulos de control, servicios de espera, servicio cliente/servidor, módulos de control simples, módulos de control complejos, y distribución en el entorno.

Este enfoque incorpora el patrón como construcción básica, y algunos símbolos gráficos para representar el estado de los servicios. Promueve la **reutilización de patrones** pero sin especificar procesos de desarrollo para/con reutilización, aunque dispone de un **entorno CASE para el lenguaje de programación C++,** denominada G++. Tampoco aborda los aspectos de gestión.

4. Proyecto RapidCIM

Smith et al. *[Smith, 92], [Smith, 94]* describen el *trabajo sobre reutilización de software de control* realizado en el Laboratorio CIM de la Universidad Estatal de Pensilvania. Dicho trabajo incorpora **tres ideas** en relación con la reutilización de software:

- **Arquitectura escalable.** Proponen una arquitectura de control jerárquica de cuatro niveles. En cada nivel se realizan tres funciones: planificación, scheduling y control, que se implementan como módulos independientes.
- **Generación automática del software de control.** Cada controlador de la arquitectura jerárquica se modela mediante una gramática fuera de contexto, y el control se realiza por medio del reconocimiento del lenguaje del sistema de fabricación definido por dicha gramática.
- **Desarrollo orientado a objetos de los componentes del sistema.** Las interfaces de las máquinas se codifican manualmente, por lo que han construido un conjunto de clases de equipamiento que proveen funciones virtuales por cada tipo de máquina. Las funciones por defecto aportan los prototipos necesarios para la generación automática de software.

El desarrollo de un sistema de control de FMS empleando las herramientas de reutilización descritas consta de **cuatro etapas:**

- Utilizar la arquitectura escalable para identificar el equipamiento, las estaciones y las células del sistema propuesto.
- Desarrollar las representaciones textuales de los movimientos de las piezas en cada estación, utilizando el generador de código automático para obtener el código de control de cada estación.
- Clasificar cada equipo y establecer las funciones de control virtuales de su contolador sobre la base de los prototipos genéricos.
- Determinar la configuración del sistema de comunicaciones y desarrollar las funciones con los prototipos de comunicación.

En este trabajo se hace especial énfasis en el **desarrollo con reutilización por medio de un conjunto de generadores de código, sin olvidar que existe un desarrollo previo para reutilizar.** No se incorporan construcciones básicas, ni notaciones gráficas nuevas, **ni se tratan aspectos de gestión.** Tampoco se especifica la aplicación de un método concreto, pero hay referencias a Booch y Rumbaugh.

5. Desarrollo de arquitecturas software

La tecnología orientada a objetos también se ha empleado en otros aspectos relacionados con los sistemas de control, como en el desarrollo de arquitecturas software, que fomenten la reutilización de diseños:

- Buschmann *[Buschmann, 92]* establece las *guías y mecanismos para construir una arquitectura reutilizable*, que soporta la reutilización de componentes existentes así como el desarrollo de nuevos componentes y diseños reutilizables.
- Koch et al. *[Koch, 95]* presentan la *arquitectura abierta para controladores de célula COSMOS*, que utiliza una plataforma de comunicaciones para garantizar la portabilidad. Esta arquitectura dispone de una herramienta CASE para acometer su implementación. En este caso, **la reutilización de diseños se acompaña de generadores de aplicación.**

6. Combinaciones con otras tecnologías

Existen otros trabajos que emplean la tecnología orientada a objetos en combinación con otras tecnologías, como por ejemplo: redes de Petri *[Fabian, 94]*.

En el caso de Wei et al. *[Wei, 96]*, la combinación es con los estándares industriales para interfaces de equipos SEMI (Semiconductor Equipment and Material International) y los protocolos para modelos genéricos de equipamiento GEM (Generic Equipment Model). Su objetivo es el desarrollo de clases de objetos de control reutilizables, que por el momento no se ha conseguido debido a la existencia de máquina de producción que no acatan GEM.

1.5 Conclusiones.

De acuerdo con la tendencia actual hacia el reciclaje y reutilización, y debido a los frecuentes cambios en los sistemas de fabricación, **el sistema de control debe ofrecer un alto grado de reutilización**, tanto en lo que respecta al hardware como al software. Para ello, desde las fases iniciales del proyecto habrá que tener muy presente este aspecto.

Las **arquitecturas funcionales de control fomentan la reutilización de diseños y componentes,** pero son insuficientes a la hora de llevar a cabo la implementación de este tipo de sistemas, requiriendo el soporte de las tecnologías de la información e ingeniería del software. En este sentido, se han analizado las distintas tecnologías de ingeniería del software involucradas en el desarrollo de aplicaciones.

Los **métodos estructurados se centran en la función y los datos** para obtener un modelo funcional. Los diseños consisten en bloques funcionales que se llaman unos a otros, representando las conexiones entre funciones mediante el paso del control. Con excepción de los sistemas de tiempo real, los métodos estructurados no realizan modelos del comportamiento del sistema en construcción.

Por el contrario, **las técnicas orientadas a objetos modelan el entorno de operación del sistema,** y contemplan los datos y funciones en un único paquete denominado objeto. Este objeto tiene uno o más estados que modelan su comportamiento, tanto en el tiempo como en respuesta a eventos internos o externos. Las conexiones entre objetos se representan mediante relaciones estructurales (herencia) o intercambio de mensajes.

Habitualmente se considera que el principal beneficio de la tecnología orientada a objetos reside en el atributo reutilizable que acompaña a los objetos. Sin embargo, esta **reutilización de código no difiere significativamente de las funciones de librería** tales como funciones C. La diferencia reside en que los objetos representan paquetes funcionales con sus estructuras de datos y operaciones disponibles sobre estas estructuras, con lo cual su empleo no precisa trabajo adicional al programador.

Pero en cualquier caso, **la reutilización de software no sucede per se.** Los diseñadores deben tener siempre en mente las ventajas de la reutilización, planificar la reutilización de lo ya existente y diseñar los nuevos componentes (en su acepción más amplia) dotándolos del atributo reutilizable.

Por lo tanto, ni la orientación a objetos es suficiente para solventar los problemas asociados con la reutilización. La única forma de conseguirlo es **la integración de la reutilización en un modelo de ciclo de vida adecuado, tal y como propugna la megaprogramación.**

No obstante, el proceso no debe ser considerado como la panacea para institucionalizar la reutilización. Es importante **establecer el énfasis apropiado en cada uno de los tres aspectos de la reutilización del software:** los **productos** producidos y consumidos por las actividades de reutilización, los **procesos** empleados para llevar a cabo la reutilización, y el **personal** involucrado.

Capítulo 2

Nuevas tendencias de Ingeniería de Software para el desarrollo del software de control de FMSs

2.1 Introducción.

Hasta la fecha, *las arquitecturas funcionales de control, los métodos estructurados, y la tecnología orientada a objetos no han solucionado los principales problemas asociados con la construcción del software de control de los sistemas de fabricación.*

La reducción del tiempo de desarrollo y puesta en marcha, así como la disminución de los costes relacionados *sólo se pueden lograr dotando a los sistemas de control de un alto grado de reutilización.*

El siguiente paso en la evolución del proceso de incorporación de la reutilización está en la **Megaprogramación.** Este enfoque de desarrollo de software está concebido desde la perspectiva de familia de productos o dominio, en contraposición a los modelos tradicionales que se centran en productos individuales. En este caso, **la reutilización está integrada en el propio ciclo de vida** y desde las primeras etapas de desarrollo.

Sobre la base de la megaprogramación, **se propone una nueva metodología para el desarrollo del software de control de FMSs** *[Llorente, 97], [Sarachaga, 97], [Sarachaga, 98].* Dicha metodología adopta *como enfoque general la reutilización del conocimiento,* y particularmente, de productos, procesos y experiencias, abandonando la idea exclusiva de considerar únicamente la reutilización de código

Aunque su principal innovación sea la incorporación de la reutilización en el propio proceso de construcción, otro gran reto es solventar el problema de los sistemas heredados. No hay que olvidar que en muchos casos el tiempo y coste de sustitución de estos sistemas pueden ser inabordables.

A lo largo del presente capítulo se aborda en detalle la Megaprogramación y se presenta una visión general de la nueva metodología para el desarrollo del software de control de FMSs.

En la **segunda sección** se define la **megaprogramación,** se describen algunas de las metodologías concebidas bajo este enfoque, y se aporta el análisis del impacto de la

Megaprogramación en la construcción de software para extraer sus beneficios potenciales.

En la **tercera sección** se presenta una **visión general de la nueva metodología,** resaltando su ciclo de vida iterativo e incremental, sus características más destacables, su enfoque de desarrollo con dos procesos independientes pero cooperantes (ingeniería de dominio e ingeniería de aplicación) para construir el software de control de FMSs, y las técnicas empleadas.

En la **cuarta sección** se exponen las **conclusiones,** destacando los beneficios que reporta la reutilización sistemática, así como la importancia de una arquitectura abstracta que capte la variabilidad del dominio.

2.2 Megaprogramación.

La megaprogramación *[SPC, 93]* es un *enfoque de ingeniería para el desarrollo de sistemas complejos a gran escala*, que prescribe la generación de software como un proceso de definición y resolución de problemas, y no como un proceso de programación sin más.

El conjunto de combinaciones problema/solución captura los aspectos comunes y variables del área del problema o dominio. Esta información se emplea con posterioridad para construir los productos específicos de la línea o familia.

El concepto de dominio se puede definir como un *área correctamente determinada donde los ingenieros manejan las propiedades comunes de los problemas soluciones del área, y las variaciones entre problemas soluciones individuales.*

El proceso de desarrollo de software bajo este enfoque y el repositorio como soporte a dicho proceso, se presentan en los siguientes apartados. También se describen las principales metodologías bajo este enfoque y se analiza el impacto de la Megaprogramación en la construcción del software.

2.2.1 Proceso de desarrollo.

El proceso de construcción del software propuesto bajo este enfoque difiere mucho de los modelos de ciclo de vida tradicionales. En lugar de realizar un proceso de desarrollo independiente para cada producto, *la megaprogramación considera el conjunto de productos que pertenecen al dominio como un todo y los productos individuales se construyen como instancias de la familia estándar.*

Esta nueva manera de construir software tiene un impacto muy significativo en el propio **ciclo de vida** *[Bandinelli, 96]*, estructurándolo en **dos procesos independientes pero cooperantes** (figura 2.1). Aunque la terminología no es común a todas las metodologías bajo este enfoque, la denominación más extendida para estos procesos es: ingeniería de dominio e ingeniería de aplicación.

- **Ingeniería de dominio.** Su objetivo es construir la familia de productos *[Conn, 97], [STARS, 96].* Con independencia de las etapas prescritas por las distintas metodologías, las principales actividades que se deben desarrollar son:
  - *Analizar y modelar el dominio.*
  - *Diseñar una arquitectura genérica para la línea de productos del dominio.*
  - *Implementar los componentes reutilizables de dicha arquitectura.*
  - *Mantener y actualizar el dominio, la arquitectura genérica y los modelos de implementación.*
- **Ingeniería de aplicación.** Su objetivo es construir instancias de la familia de productos *[STARS, 93].* Cuando la megaprogramación está madura e institucionalizada, este proceso se reduce a:
  - *Identificar el dominio, sus productos y procesos asociados.*
  - *Adaptar y transformar los productos existentes.*
  - *Generar la aplicación y componerla.*
  - *Realizar el diseño y la codificación de los requisitos no contemplados.*
  - *Integrar los componentes y realizar las pruebas.*

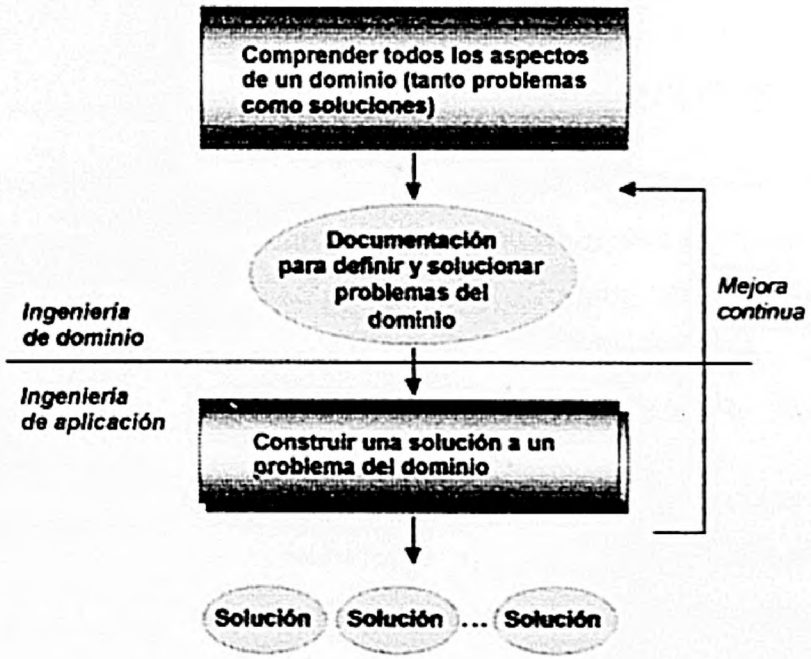


Figura 2.1: Megaprogramación.

Por otra parte, la ingenieria de dominio e ingenieria de aplicación también pueden ser consideradas como tecnologías *[Gerhart, 94]:*

- **Ingeniería de dominio.** Las tecnologías que incluye se clasifican en tres grandes categorías:
  - *Modelado de producto.* Construcción de los modelos formales e informales del espacio del problema y la solución del dominio, abarcando requisitos, arquitecturas, diseños, codificaciones y las relaciones entre ellos.
  - *Modelado de proceso.* Representación de la información que describe todos los elementos del proceso correspondiente a un sistema y sus relaciones.
  - *Implementación de arquitecturas.* Diseño de una arquitectura genérica para una línea de productos del dominio particular y el desarrollo de componentes reutilizables para dicha arquitectura.
- **Ingeniería de aplicación.** Sus tecnologías también se clasifican en tres grandes categorías, coincidiendo dos de ellas con las ya comentadas para la ingeniería de dominio:
  - *Modelado de producto.*
  - *Modelado de proceso.*
  - *Instanciación de la aplicación.* Construcción de una instancia de la familia de productos a partir de los elementos generados por la ingeniería de dominio.

2.2.2 Repositorio.

Un elemento muy importante en el proceso de desarrollo de software bajo el enfoque de Megaprogramación es el repositorio o librería, porque actúa como *integrador de los procesos de ingeniería de dominio e ingeniería de aplicación.*

Una librería específica del dominio proporciona muchos de los servicios de infraestructura claves para soportar el proceso dinámico de construcción y adaptación, así como la definición y mejora de los componentes de la línea de productos.

Las **librerías reutilizables actuales** *[STARS, 96]* se clasifican en:

- **Librerías basadas en componentes.** Almacenes software en los que el elemento central es el componente.
- **Librerías basadas en el modelo.** El elemento principal es el modelo del dominio o arquitectura genérica. La organización de estas librerías refleja las relaciones existentes entre los componentes reutilizables.

Una librería específica del dominio típica (sin tener en cuenta si es basada en el componente o en el modelo) *contiene un modelo del dominio que puede incluir representaciones de los requisitos genéricos del dominio, una arquitectura genérica, los componentes implementados, la documentación de los componentes y las experiencias obtenidas previamente.*

Además, *se precisan un conjunto de procedimientos que aseguren la disponibilidad y accesibilidad de la librería y de sus elementos para su posterior uso.*

Pero las librerías no son una solución de reutilización, sino simplemente una **herramienta de apoyo para realizar la reutilización en el dominio.** Por este motivo, toda librería debe ser *objeto de un proceso de mejora continuo,* muy influenciado por el feedback entre ingenieros de dominio e ingenieros de aplicación.

2.2.3 Principales metodologías de desarrollo de software bajo el enfoque de la Megaprogramación.

2.2.3.1 SFLC (Software-First Life Cycle).

El SFCL *[SFLC, 90]* de IBM describe un ciclo de vida, fruto del estudio y evaluación de los ciclos de vida existentes y de las tecnologías emergentes que tienen el potencial para mejorar significativamente el proceso de desarrollo de sistemas.

Los **conceptos fundamentales** que incorpora son:

- **Prototipado.**
- **Reutilización en todas las fases del proceso.**
- **Ingeniería concurrente.**
- **Adaptación del proceso al sistema concreto.**
- **Comunicación abierta con los clientes.**
- **Equipos multidisciplinares.**
- **Análisis del dominio.**
- **Independencia del hardware.**

El ciclo de vida que propone SFLC se estructura en cinco fases, presentando todas ellas una fuerte interacción con el repositorio (figura 2.2):

- **Análisis preliminar del sistema.** El principal propósito de esta etapa es la comprensión del sistema y de los requisitos de usuario, con el detalle suficiente como para poder construir un prototipo inicial en la siguiente etapa. También se planifica el proyecto, se establece el entorno de desarrollo y se identifican nuevas tecnologías.
- **Arquitectura del sistema.** El prototipo inicial del sistema y las subsecuentes ampliaciones del mismo se construyen a partir de componentes reutilizables existentes en el repositorio. En aquellos casos que se requieran nuevos componentes o tecnologías, las solicitudes correspondientes se transmiten a la siguiente fase.
- **Desarrollo de software.** El desarrollo de nuevos componentes software se realiza mediante prototipos incrementales, los cuales se almacenan en el repositorio tras la superación de las pruebas asociadas.
- **Producción.** El prototipo del sistema se convierte en el producto final, es decir, se pasa de un sistema ejecutable funcionalmente a una sistema definido, especificado, desarrollado, documentado, probado y entregado.

- **Operación del sistema y soporte.** La identificación de nuevos requisitos funcionales son nuevas entradas a la fase de Análisis preliminar de la siguiente versión del sistema.

Finalizadas las fases de análisis y de arquitectura se realizan exhaustivas revisiones. La primera permite comprobar que se comprende el sistema y los requisitos de usuario, mientras que la segunda garantiza el acuerdo sobre las tareas necesarias para obtener el sistema final.

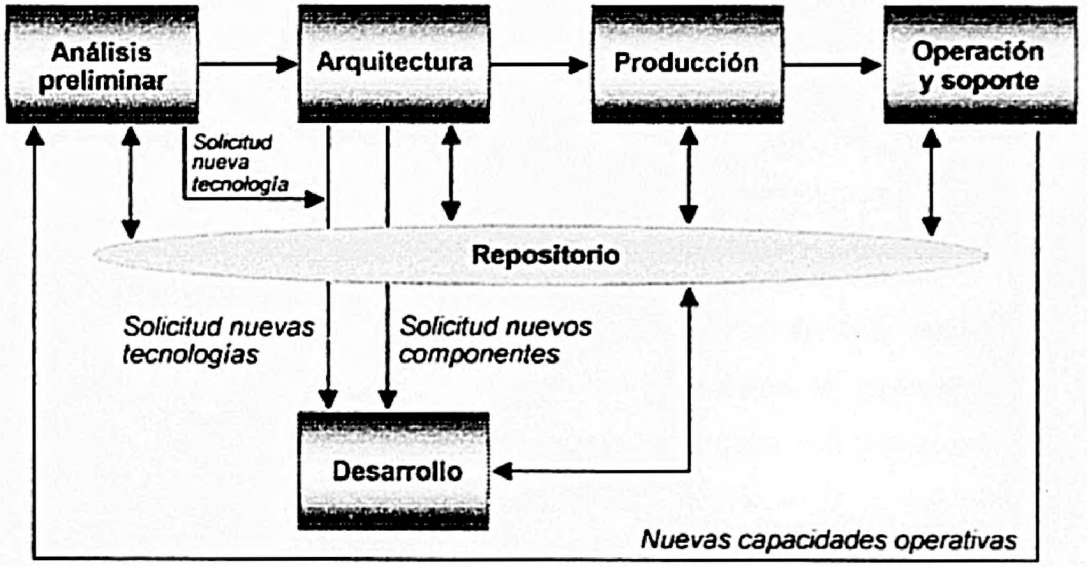


Figura 2.2: Ciclo de vida SFLC.

2.2.3.2 CFRP (Conceptual Framework for Reuse Processes).

Desarrollado bajo el auspicio del programa STARS[16] (Software Technology for Adaptable, Reliable Systems), el CFRP *[STARS, 93a]* define un contexto para los procesos de desarrollo de software afines a la reutilización, sus relaciones, y cómo se integran entre si y con procesos no relacionados con la reutilización.

[16] El objetivo del programa STARS es el desarrollo de procesos de construcción de sistemas y de las nuevas tecnologías de apoyo necesarias, de cara a poder afrontar la construcción de grandes y complejos sistemas con gran calidad, en menos tiempo y con unos costes más bajos.

El propósito es definir modelos de proceso para ciclos de vida orientados a la reutilización, que puedan ser adaptados según las necesidades de las organizaciones.

El CFRP consta de **dos procesos duales interconectados,** denominados Gestión de la reutilización e Ingeniería de la reutilización, que describen los distintos patrones de actividades inherentes a los aspectos de gestión y organización, y a los aspectos de ingeniería de producto, respectivamente.

- **Gestión de la reutilización.** Patrón cíclico de actividades que acomete el establecimiento y mejora continua de las tareas afines a la reutilización dentro de una organización, fomentando el aprendizaje como mecanismo institucional para el cambio. La gestión de la reutilización engloba *tres familias de procesos:*
  - *Procesos de planificación.* Establecen los objetivos y estrategias de reutilización dentro y entre dominios, planifican el conjunto de proyectos reutilizables relacionados, y planifican la infraestructura necesaria para llevar a cabo el proyecto y su evolución.
  - *Procesos de representación.* Gestionan los proyectos de reutilización activos, regulan el día a día de dichos proyectos, y garantizan la disponibilidad de la infraestructura.
  - *Procesos de aprendizaje.* Evalúan los resultados de los proyectos en relación con los objetivos locales y globales, y hacen recomendaciones para mejorar las capacidades de reutilización.
- **Ingeniería de la reutilización.** Patrón en cadena de actividades referentes al desarrollo de productos para reutilizar, su gestión y su reutilización. Las familias de procesos que identifica son las siguientes:
  - *Procesos de creación.* Construyen y evolucionan los modelos y elementos del dominio, incluyendo requisitos, arquitecturas, generadores de aplicación, y componentes software.
  - *Procesos de gestión.* Adquieren, describen, evalúan, organizan y gestionan los elementos proporcionados por la familia de procesos de creación, y suministran servicios para fomentar y facilitar la reutilización.

- *Procesos de utilización.* Reutilizan los elementos dispuestos por la familia de procesos de gestión, es decir, los identifican, seleccionan, adaptan e integran para construir la aplicación.

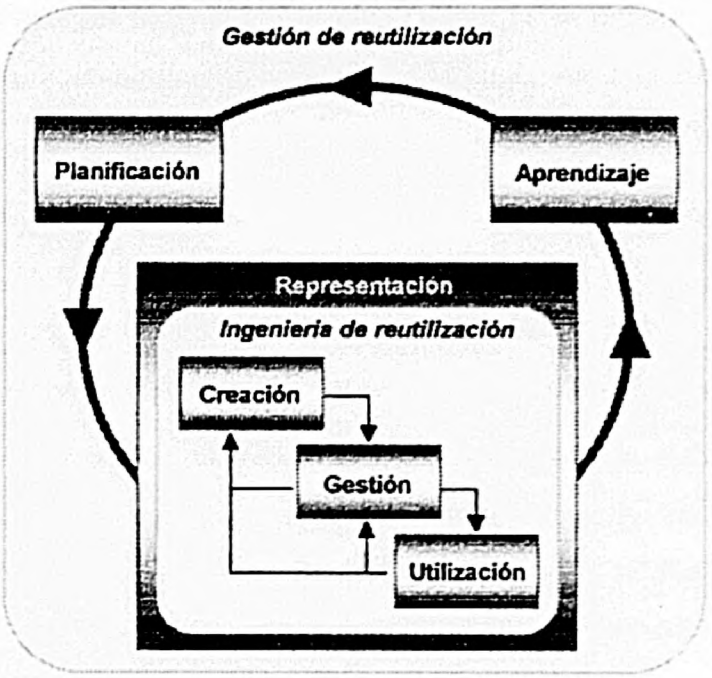


Figura 2.3: Configuración canónica del CFRP.

La separación en dos procesos permite distinguir claramente entre los dos aspectos de las actividades relacionadas con la reutilización, pero también permite realizar configuraciones flexibles de los procesos dependiendo de los niveles de planificación, estructuras organizativas y patrones de interacción. No obstante, la *configuración canónica* (figura 2.3) ofrece la imagen más intuitiva del CFRP, donde las actividades de ingeniería están controladas directamente por las actividades de gestión.

2.2.3.3 Synthesis.

Synthesis *[SPC, 93a]* es la metodología de SPC (Software Productivity Consortium) para la *construcción de sistemas software como instancias de una familia de productos.* Al aprovechar las similitudes entre productos de la familia, se elimina trabajo redundante.

Este enfoque sistemático para el desarrollo de software se fundamenta en la *gestión de los recursos no sólo para abordar las necesidades inmediatas de los clientes, sino también en la investigación de la capacidad futura.* Aunque los **cuatro fundamentos básicos** son:

- **Familia de productos.**
- **Procesos iterativos.**
- **Especificaciones como modelo abstracto.**
- **Reutilización basada en la abstracción.**

El proceso persigue dos objetivos independientes pero relacionados: *producir y entregar sistemas software, e incrementar la productividad, calidad, gestión y responsabilidad de la producción y entrega del software.* Para acometer efectivamente estos intereses, **Synthesis integra dos procesos iterativos:**

- **Ingeniería de aplicación.** Su objetivo es equivalente al de un ciclo de vida convencional: construcción de aplicaciones y entrega de las mismas. La diferencia estriba en que la ingeniería de aplicación tiene un proceso definido y apoyado por componentes reutilizables.

- **Ingeniería de dominio.** Como soporte a la ingeniería de aplicación, construye los componentes reutilizables y define su proceso automatizándolo en la medida de lo posible. Las *actividades* (figura 2.4) que comprende son:
  - *Gestión del dominio.* Planificación, supervisión y control del esfuerzo necesario para realizar la Ingeniería de dominio.
  - *Análisis del dominio.* Especificación de requisitos para la familia de productos identificando la variabilidad entre productos individuales, desarrollo de una arquitectura genérica y descripción del proceso para la ingeniería de aplicación.
  - *Implementación del dominio.* Construcción de los componentes reutilizables y del soporte automático al proceso de ingeniería de aplicación.
  - *Apoyo a proyectos.* Asistencia en los proyectos individuales para usar de forma efectiva los elementos generados para el dominio.

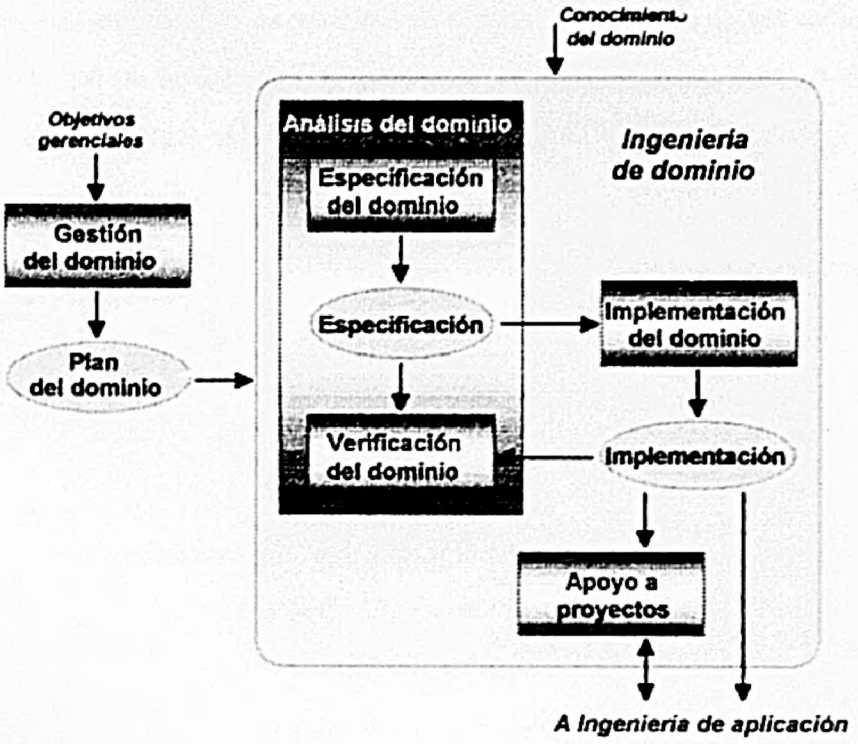


Figura 2.4: Proceso de Ingeniería de dominio Synthesis.

Modelo de implantación

No obstante, la adopción de un proceso Synthesis precisa su adaptación a la realidad de la organización. Para ello, *SPC propone el Modelo de Capacidad de Reutilización* (RCM – Reuse Capability Model) *[SPC, 93b]* que permite derivar la definición del proceso más apropiado, e incluye un modelo incremental de implantación con cuatro etapas:

- **Oportunista.** El esfuerzo reside en la construcción de componentes reutilizables para satisfacer las necesidades inmediatas de los proyectos actuales.
- **Integrada.** El proceso de desarrollo integra la reutilización, y los componentes son desarrollados para uso múltiple en las necesidades actuales.
- **Aventajada.** Los componentes se construyen considerando las necesidades presentes y futuras de la línea de productos.
- **Anticipada.** El potencial de reutilización es una consideración prioritaria en el modo de afrontar los problemas.

2.2.3.4 SCAI (Space Command and control Architectural Infraestructure).

El proyecto SCAI *[Bristow, 95]* de las Fuerzas Aéreas Norteamericanas, aprovecha los avances tecnológicos en Megaprogramación obtenidos por STARS, y define un *proceso de línea de producto sobre la base de resultados y métodos prácticos que simplifican la ingeniería de dominio e ingeniería de aplicación.*

Para acometer este empeño, **SCAI integra tres tecnologías:**

- **Análisis y diseño orientado a objetos de Booch.** Las técnicas de Booch permiten abstraer las vistas lógicas de los sistemas y dominios iterativamente.
- **Modelo de proceso para Ada[17].** Aborda las actividades de diseño para sistemas de control y comandos de forma iterativa, y proporciona representaciones intermedias de la arquitectura física del sistema.
- **Cleanroom[18].** Aporta un enfoque riguroso para el desarrollo de software cuando las vistas de la arquitectura (lógica y física) están ya determinadas

De la combinación de las características de estas tecnologías resulta un conjunto de actividades que proporcionan una descripción a alto nivel del proceso integrado (figura 2.5). Las actividades identificadas para cada **ciclo de vida** son las siguientes:

- **Ingeniería de aplicación:**
  - *Acopio de requisitos del sistema.* Obtención de la información necesaria para delimitar el ámbito del modelo de objetos particular.
  - *Especificación del sistema.* Definición de los requisitos al nivel de sistema.
  - *Preparación del Modelo de objetos.* Las abstracciones o clases del dominio se detallan y validan.

---

[17] El Modelo de proceso para Ada se caracteriza por el empleo del lenguaje Ada, un proceso de construcción iterativo, unas revisiones en base a demostraciones, y unos generadores de código basados en una arquitectura.

[18] Cleanroom es un método formal que proporciona un enfoque integrado estricto desde la preparación de la especificación hasta el desarrollo y certificación del software, brindando al ingeniero un control riguroso sobre el proceso.

- *Desarrollo del Modelo de arquitectura.* Elección de la arquitectura tras el estudio de las alternativas para el modelo de objetos consolidado.
- *Desarrollo de software incremental.* Construcción de los distintos componentes identificados en la arquitectura.
- *Certificación del software.* Esta actividad verifica que el software exhibe el comportamiento que está definido en la especificación.

- **Ingeniería de dominio:**
  - *Acopio de requisitos del dominio.* Especificación de los requisitos generales para los sistemas que pertenecen al dominio.
  - *Generalización del Modelo de objetos.* Esta actividad realiza la abstracción del modelo de objetos para abarcar todo el dominio, contemplando también la inclusión de nuevos requisitos o de conclusiones obtenidas en aplicaciones particulares.
  - *Gestión del dominio.* Gestión de todos los recursos necesarios para mantener la línea de productos asegurando la máxima reutilización. También se encarga de planificar la evolución de la línea de productos, especialmente con la introducción de los nuevos avances tecnológicos.

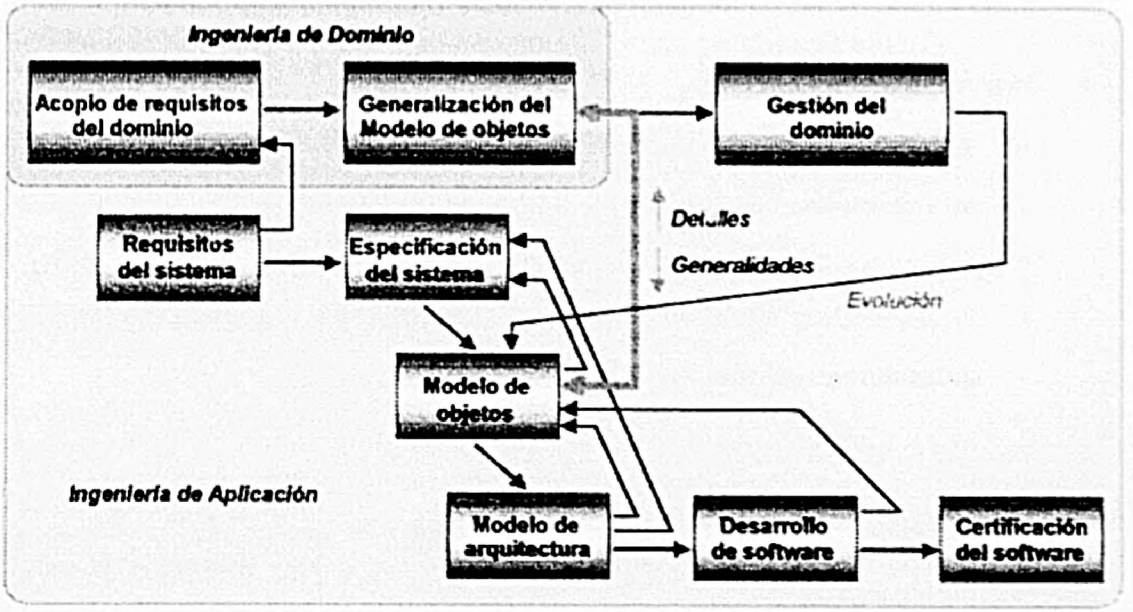


Figura 2.5: Modelo del proyecto SCAI.

2.2.3.5 Líneas de productos ESC-Hanscom (Electronic Systems Center of Hanscom).

El Centro de Sistemas Electrónicos (ESC) de la base de las Fuerzas Aéreas en Hanscom (Massachusetts) ha adoptado una *estructura organizativa de líneas de productos* para afrontar el desarrollo de sistemas software[19] *[Cohen, 95]*. Esta recomendación fue fruto de un estudio previo *[Cohen, 96]*, en el cual también se identificaron las líneas de productos que se debían implementar.

El **soporte al proceso de construcción** de los sistemas está a cargo de tres grupos u organizaciones:

- **Grupo de Arquitectura de Sistemas (SAG – System Architectures Group).** Este grupo se encarga de la definición de la arquitectura para la línea de productos, pero también de la adaptación de dicha arquitectura al caso de una aplicación específica.
- **Centros de Ingeniería de la Línea de Productos (PLEC – Product Line Engineering Centers).** Cada PLEC define y evoluciona la arquitectura de su línea de productos con el SAG, desarrolla componentes y realiza la gestión de configuración. Además, construye prototipos y el plan de desarrollo para sistemas particulares.
- **Grupo de Soporte de Componentes de la Líneas de Productos (PLAS – Product Line Asset Support Group).** El PLAS soporta la reutilización de componentes mediante su identificación, empaquetamiento y cualificación, garantizando el éxito de la utilización de los mismos en/entre líneas de productos.

El **proceso para desarrollar sistemas** con este enfoque de línea de productos distingue las siguientes actividades:

---

[19] Sistemas inteligentes, de comunicaciones, control o comandos con carácter militar.

- **Adaptación de la arquitectura de la línea de productos.** La arquitectura de un sistema específico se obtiene mediante la adaptación de la arquitectura de la línea de productos basándose en los requisitos de usuario.
- **Integración de los componentes.** Partiendo de la arquitectura obtenida en el paso anterior, el nuevo sistema se construye integrando componentes (adaptados o no) de la línea de productos u otras líneas, y si fuera necesario, componentes comerciales y/o de nuevo desarrollo.
- **Cualificación, pruebas operativas y evaluación del sistema.**

La arquitectura del nuevo sistema y aquellos componentes modificados, comerciales o de nueva construcción, constituyen nuevos elementos para emplear en futuros desarrollos. De aquí se desprende la *importancia de las actividades de mantenimiento y gestión de configuración tanto de versiones de sistemas como de componentes.*

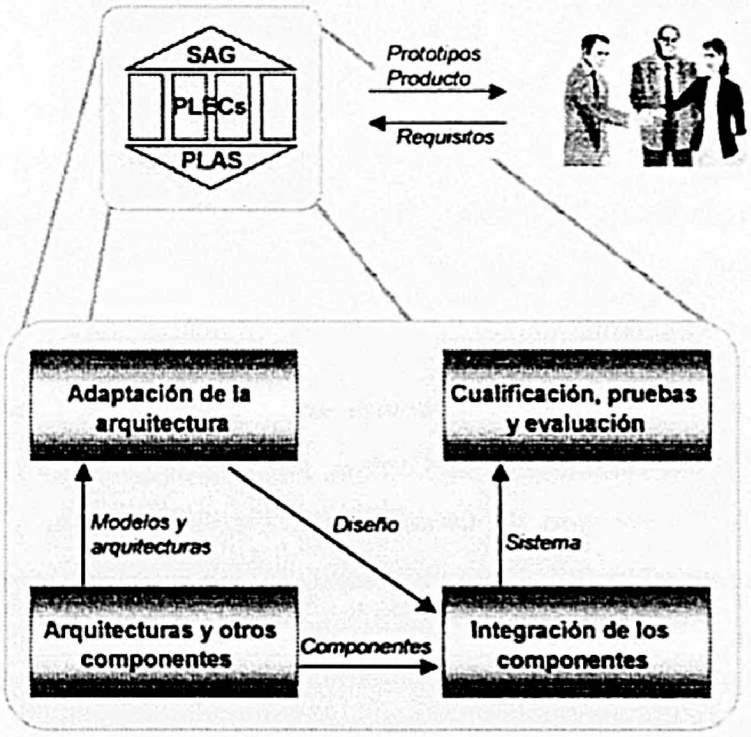


Figura 2.6: Líneas de productos ESC-Hanscom.

2.2.4 Análisis del impacto de la Megaprogramación en la construcción de software.

Las metodologías descritas, consideradas bajo el enfoque de la Megaprogramación, **incluyen la reutilización en el propio proceso de construcción del software** y desde su concepción; incluyen el desarrollo para reutilizar (ingeniería de dominio), el desarrollo con reutilización (ingeniería de aplicación) y los aspectos de gestión de los repositorios. La mayoría de ellas no establecen técnicas concretas de desarrollo, aunque *alguna sí recomiende explícitamente el empleo de la tecnología orientada a objetos.*

A diferencia de los modelos de ciclo de vida tradicionales, concebidos para desarrollar productos software individuales, la Megaprogramación es un enfoque de desarrollo de software desde la perspectiva de familia de productos o dominio. Además, **se enfatiza la reutilización de arquitecturas más que la de componentes.**

Este nuevo enfoque representa un cambio cultural al considerar la **construcción de software como una disciplina de fabricación.** Como tal, el impacto que provoca en la industria del software afecta a varios niveles:

- **Planificación estratégica.** La identificación de similitudes y variaciones centra la planificación estratégica en:
  - *Desarrollar la infraestructura (tecnología, personal, componentes reutilizables y procesos) para construir los productos en el menor tiempo y al menor coste.*
  - *Prever las tendencias tecnológicas para la línea de productos.*
  - *Identificar las posibles ventajas competitivas de la línea de productos (productos básicos en poco tiempo, a bajo coste, etc.).*
- **Planificación de los proyectos.** Los procesos específicos del dominio aportan mayor precisión a la planificación de los proyectos sin más que:
  - *Identificar las tareas del dominio que se deben realizar para el producto concreto, y las estimaciones de recursos típicas.*

- *Identificar las tareas necesarias para acometer las variaciones del cliente, y si dichas tareas requerirán algún desarrollo especial.*

- **Ejecución de los proyectos.** La arquitectura de dominio, sus componentes probados y la documentación asociada simplifican la ejecución del proyecto al:
  - *Eliminar ciertas tareas de desarrollo, tales como la definición de una arquitectura.*
  - *Reducir la tarea de análisis a comprobar cómo encajan los componentes existentes en las necesidades del cliente.*
  - *Reducir las tareas de pruebas.*

- **Empresa.** El hecho de centrarse en un problema concreto y sus soluciones permite a la empresa:
  - *Obtener una imagen corporativa.*
  - *Romper las barreras culturales.*
  - *Rediseñar los procesos de gestión.*

En la medida que la Megaprogramación esté más utilizada en la construcción de software y maduren las líneas de productos desarrolladas bajo este enfoque, se obtendrán importantes beneficios tales como:

- *Consolidación de los recursos y competencias de áreas de negocio claves.*
- *Incremento de la calidad porque se emplean componentes probados.*
- *Construcción de componentes adaptables a las necesidades de varios usuarios.*
- *Reducción del número de componentes, minimizando los costes de desarrollo repetitivos y totales.*
- *Reducción del riesgo en el rendimiento del sistema, ya que se conoce el rendimiento de los componentes.*
- *Mejora del tiempo de producción gracias a la reutilización de la tecnología, los diseños y los componentes.*

- *Incremento de la interoperabilidad debido a la reutilización de arquitecturas comunes, interfaces y protocolos.*
- *Reducción de los requisitos de formación por motivo de la similitud entre componentes.*

Por el momento, *ninguna de las metodologías descritas en los apartados anteriores tiene gran difusión en la comunidad de desarrollo de software.* No obstante, se pronostica **Synthesis como una de las metodologías más prometedoras a corto plazo,** por el éxito obtenido en distintas aplicaciones con carácter industrial, entre otras:

- Determinar la variabilidad y elaborar el proceso de decisión para un dominio de un simulador de vuelo genérico en Boeing *[Freeman, 92]*.
- Crear el dominio para el software de comunicaciones CCM MIL-STD-1553B en Rockwell International *[O'Connor, 94]*.
- Establecer distintas líneas de productos en Syseca *[Potter, 97]*.

Aplicando los principios básicos del nuevo enfoque para la construcción de software, y combinando las principales ventajas de las metodologías estudiadas, especialmente de Synthesis, **se ha concebido una nueva metodología específica para el dominio del software de control de FMSs.**

Esta nueva metodología *también propone el desarrollo de software como una disciplina de fabricación,* con el impacto que ello conlleva. Los beneficios a medio y largo plazo son los esperados para líneas de productos maduras, aunque a corto plazo se persigue la reducción del tiempo y coste de puesta en marcha de este tipo de instalaciones.

Una visión general de esta nueva metodología con reutilización sistemática para el software de control de FMSs se presenta en la siguiente sección.

2.3 Visión general de la nueva metodología.

A continuación se describe el ciclo de vida de la nueva metodología, se definen sus características más reseñables, y se presentan su proceso de construcción del software y las técnicas empleadas.

2.3.1 Ciclo de vida.

La nueva metodología propone un proceso de desarrollo evolutivo que permite además resolver el problema de los sistemas heredados, los cuales no pueden ser reemplazados por los costes asociados. Este carácter evolutivo es inherente al carácter iterativo e incremental de su ciclo de vida:

- **Iterativo.** Comprende un refinamiento sucesivo, en el cual se aplica la experiencia y los resultados de cada revisión a la siguiente iteración del análisis y diseño.
- **Incremental.** Cada paso a través del ciclo análisis/diseño/evolución tiende hacia un refinamiento gradual, que converge en una solución acorde con los requisitos potenciales de un grupo de usuarios.

2.3.2 Características.

Las características más reseñables de este modo de crear y mantener el software de control para FMSs son reutilización, visión de dominio, arquitectura genérica para la familia de productos, guías para el proceso de desarrollo y librerías de elementos reutilizables, definidas a continuación:

- **Reutilización.** Esta sistematización para el desarrollo y evolución del software consiste en derivar nuevos sistemas o modificar los previos basándose en elementos existentes, en lugar de crear dichos sistemas desde su inicio. *Los elementos reutilizables son el kernel de la nueva metodología* y requieren procesos para su creación, gestión y empleo. *Estos elementos incluyen no sólo*

componentes de código comúnmente asociados con la reutilización, sino también otro tipo de información, especialmente:

- *Productos software:* especificaciones, arquitecturas, diseños, procedimientos de prueba, etc.
- *Conocimiento del área de aplicación:* modelos, diccionarios de datos, algoritmos, etc.
- *Definición de procesos* para gestionar librerías de componentes, desarrollar sistemas de aplicación, etc.
- *Razonamiento* para añadir características, servicios, objetos y/o algoritmos en un sistema, seleccionar una arquitectura o diseño, etc.

- **Visión de dominio.** *Los elementos reutilizables, los procesos de desarrollo y la tecnología de soporte son a medida de la familia de productos* constituida por el software de control de FMSs. Como ocurre generalmente, este dominio tiene un ámbito amplio y engloba a su vez a otros dominios, que pueden ser únicos en el área de estudio (dominios verticales) o comunes a varios dominios (dominios horizontales).

- **Arquitectura genérica para la familia de productos con interfaces estándares.** Proporciona la guía para crear los componentes y construir las instancias de la familia de productos. En este sentido, es el elemento crítico para la reutilización del dominio, *cuya efectividad depende de* varios factores entre los que destacan: *la madurez del dominio y la inversión destinada a crear elementos reutilizables.*

- **Conjunto de guías que establecen el proceso de desarrollo.** La construcción del software se realiza según procesos repetibles bien definidos que son objeto de mejora continua. En la medida de lo posible *es recomendable el uso de entornos de ingeniería de software asistidos por ordenador,* que proporcionan una definición automatizada de los procesos y guían en su aplicación.

- **Librerías de elementos reutilizables.** Repositorios diseñados, construidos y mantenidos para almacenar los elementos reutilizables generados para el

dominio. *Disponen de mecanismos para la recuperación* de dichos elementos *y están sujetas a procesos de mejora continua.*

2.3.3 Proceso de desarrollo.

El nuevo enfoque de construcción del software de control de FMSs *comprende dos procesos unitarios, independientes pero cooperantes:* **ingeniería de dominio e ingeniería de aplicación** (figura 2.7), descritos en el siguiente capítulo.

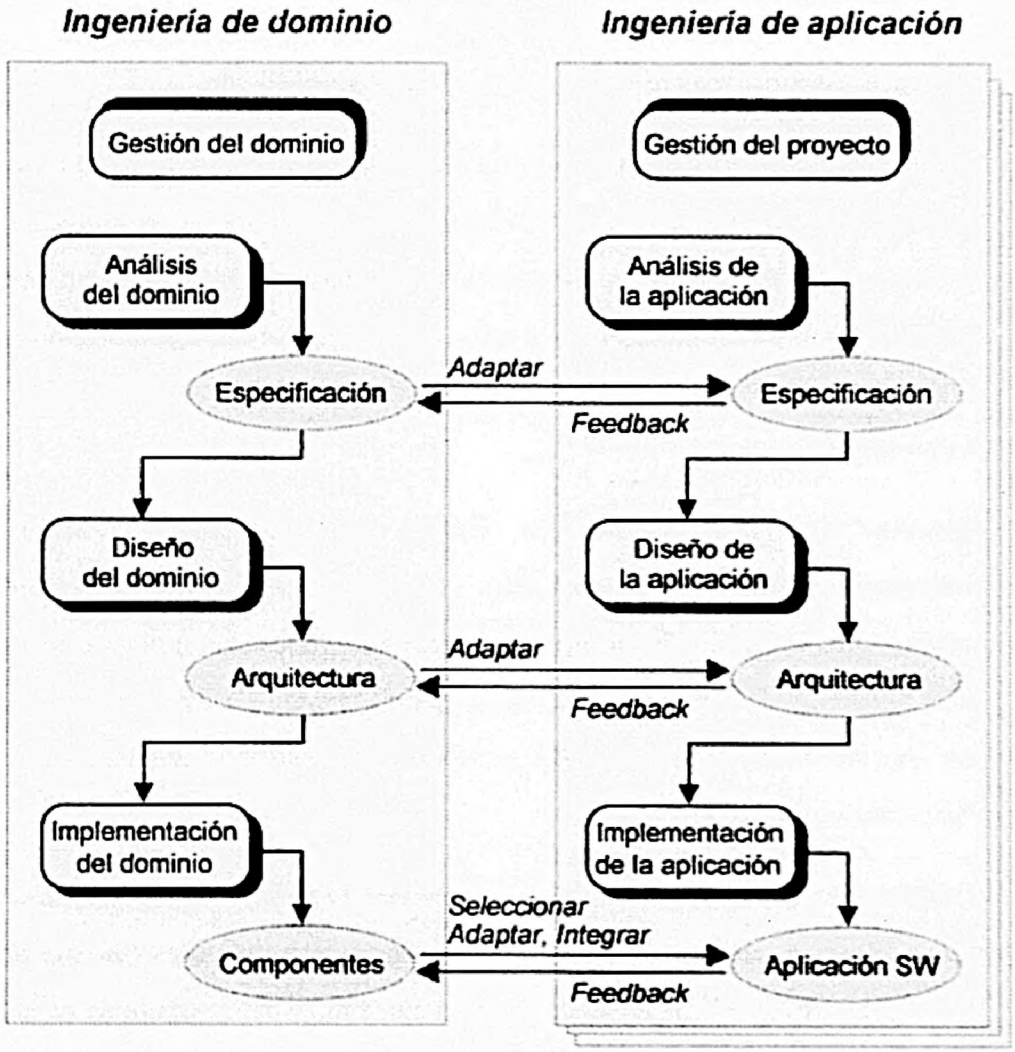


Figura 2.7: Proceso de desarrollo del software.

El motivo principal de esta descomposición radica en la disparidad de los objetivos que persiguen estos procesos:

- *La ingeniería de dominio se centra en el desarrollo de una familia de productos,* dejando abiertas las decisiones relativas a los requisitos específicos del producto.
- *La ingeniería de aplicación deriva un producto individual* a partir de la familia de productos, y de acuerdo con los requisitos concretos del cliente.

La cooperación entre sendos procesos es bilateral (figura 2.8) a través de un mecanismo de realimentación: la ingeniería de dominio proporciona componentes y guías a la ingeniería de aplicación, y esta última le proporciona *feedback* para mejorar dichos elementos.

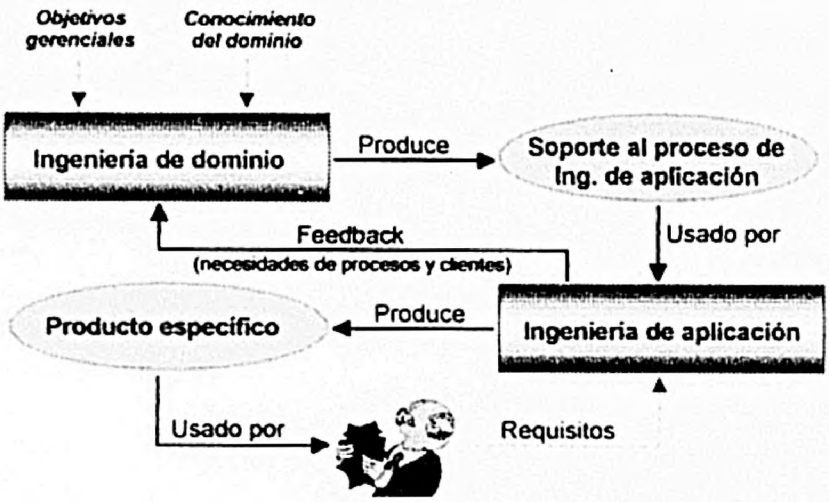


Figura 2.8: Cooperación entre la ingeniería de dominio y la ingeniería de aplicación.

2.3.4 Técnicas.

En cuanto a las técnicas, se emplea **descomposición funcional, orientación a objetos y análisis de eventos.** La descomposición funcional permite identificar un conjunto de módulos que son los objetos al nivel del dominio, los cuales encapsulan su funcionalidad y sus datos.

Por lo tanto, *el dominio se representa en un modelo estático de objetos, denominado arquitectura genérica del dominio,* que consta de un conjunto de objetos que se

comunican. De este modo, se resuelve en gran medida el problema de integrar módulos funcionales existentes en un nuevo sistema.

Pero la representación de la funcionalidad del dominio precisa un análisis más profundo de los mensajes que se intercambian los objetos de aplicación. *El modelo del comportamiento dinámico del dominio se realiza mediante el análisis de eventos*, que captura la comunicación entre objetos en los diagramas de flujo de mensajes.

El modelo dinámico proporciona la especificación de mensajería de los distintos módulos, cuyas arquitecturas genéricas se obtienen mediante descomposición funcional.

En implementación, los lenguajes de programación orientados a objetos ofrecen mayores ventajas para construir componentes adaptables[20], pero existen otras técnicas igualmente buenas soportadas por otros lenguajes.

---

[20] Los lenguajes orientados a objetos soportan mecanismos que fomentan la reutilización: herencia, polimorfismo, encapsulación, abstracción, asociación, agregación.

2.4 Conclusiones.

La Megaprogramación propugna la integración de la reutilización en el propio modelo de ciclo de vida para obtener todos los beneficios de una reutilización sistemática. Todas las metodologías presentadas dentro de este enfoque, incluida la propuesta, así lo constatan.

La metodología propuesta establece el esquema de construcción de software reutilizable para FMSs. Este esquema determina un conjunto de pasos a seguir que garantizan la reutilización del software construido, y se analizan detalladamente en el siguiente capítulo.

No obstante, **lo más importante es tener una arquitectura abstracta que acoja la variabilidad del dominio y una implementación que permita su instanciación.** Por ello, dadas las características intrínsecas del dominio del software de control de FMSs, *se debe emplear descomposición funcional, orientación a objetos y análisis de eventos en las etapas de análisis y diseño.*

En lo que respecta al área de fabricación, la posibilidad de utilizar los mismos componentes en distintas aplicaciones o de adaptar algunos a las necesidades específicas con un mínimo esfuerzo, sin tener que desarrollar íntegramente cada aplicación, **constituye un avance importante en la construcción del software de control para FMSs.**

Como colofón, indicar que **la metodología propuesta se puede generalizar para cualquier familia de productos sin más que obviar los aspectos específicos del dominio aquí tratado.**

Capítulo 3

Metodología con reutilización sistemática para el desarrollo del software de control de FMSs

3.1 Introducción.

En la visión general de la nueva metodología para el desarrollo del software de control de FMSs, **quedó de manifiesto la necesidad de una arquitectura lo suficientemente genérica** como para representar la familia de productos con su variabilidad, y que contemple, además, la posibilidad de incluir sistemas existentes construidos sin tener presente la reutilización.

Tras el análisis de distintas arquitecturas funcionales, realizado en el primer capítulo del presente trabajo, *se propone una arquitectura basada en la propuesta por Robotiker [Robotiker, 92],* modificada completándola con los aspectos derivados de la propuesta por Adiga *[Adiga, 93].*

Pero no sólo la arquitectura está respaldada por un análisis previo. En el caso de la nueva metodología, *el análisis de los métodos estructurados, la tecnología orientada a objetos y la megaprogramación (capítulos 1 y 2), ha permitido extraer aspectos idóneos para el desarrollo de este tipo de software.* En este sentido, se destacan:

- *Los diagramas de flujo de mensajes utilizados por Adiga [Adiga, 93]* para capturar la comunicación entre procesos. Se emplean ahora para describir el comportamiento dinámico del sistema, y obtener las especificaciones de mensajería de los módulos.
- *La actividad de "Apoyo a proyectos" identificada en la metodología Synthesis [SPC, 93a].* Se descompone y traduce en la tarea "Soporte a la ingeniería de aplicación" que se adjunta en todas y cada una de las actividades (análisis, diseño, implementación y gestión) del proceso de ingeniería de dominio de la nueva metodología.

También se incorporan aspectos derivados de los métodos estructurados. De hecho, *se emplean descomposición funcional junto con orientación a objetos y análisis de eventos en las etapas de análisis y diseño, y la implementación de algunos módulos se lleva a cabo con programación estructurada.*

Todos estos aspectos se detallan a lo largo de este capítulo, que presenta los dos procesos asociados con la construcción del software de control de FMSs, y la construcción del repositorio como nexo de unión entre ambos procesos.

En la **segunda sección** se explica el proceso de **ingeniería de dominio,** definiendo las distintas actividades (análisis, diseño, implementación y gestión del dominio), sus tareas, y los productos generados en cada una de ellas. También se presenta la construcción del repositorio por ser una responsabilidad más de este proceso.

En la **tercera sección** se describe el proceso de **ingeniería de aplicación,** destacando que los procesos a seguir en el desempeño de sus actividades (análisis, diseño, implementación y gestión del proyecto) están especificados por el proceso de ingeniería de dominio.

En la **cuarta sección** se exponen las **conclusiones**, resaltando los beneficios que reporta la reutilización sistemática que incorpora la metodología propuesta, y cómo afecta positivamente al área de la fabricación flexible.

3.2 Ingeniería de dominio.

Este proceso consiste en la *creación de la familia de productos constituida por el software de control para FMSs*, y en ciertos casos, la construcción de herramientas automáticas para los procesos asociados con el desarrollo de componentes concretos. El resultado es un conjunto de productos obtenidos mediante una serie de actividades.

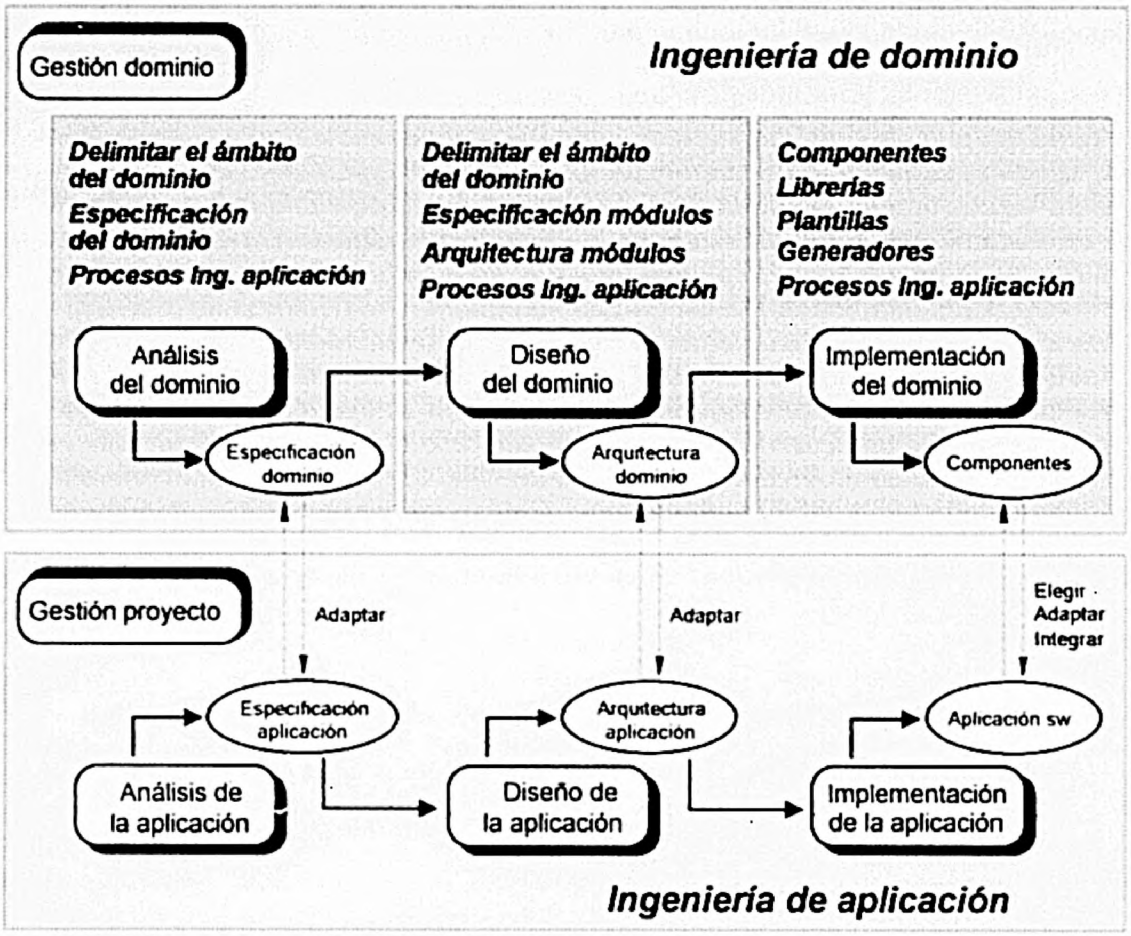


Figura 3.1: Actividades y productos de la ingeniería de dominio.

Los principales productos generados durante este proceso son los siguientes: *una descripción de los sistemas incluidos dentro de esta familia de productos, una solución genérica para desarrollar tales sistemas, una arquitectura genérica para cada uno de los módulos que componen la solución genérica, una serie de componentes reutilizables, y un conjunto de procedimientos para la ingeniería de aplicación.*

La descripción de los problemas pertenecientes al dominio se recoge en la definición y especificación de la familia de productos, mientras que la solución a dichos problemas se expresa en forma de diseño de alto nivel o arquitectura genérica.

La arquitectura genérica del dominio constituye el elemento clave del ciclo de vida de la ingeniería del dominio: divide el sistema en componentes discretos no solapados y define interfaces explícitas entre los mismos. De este modo, establece la relación entre los requisitos de la familia de productos y los componentes reutilizables construidos sobre la base de sus propias arquitecturas genéricas.

Los pasos a seguir para adaptar la arquitectura genérica del dominio a las necesidades de un proyecto concreto se definen en un conjunto de procedimientos que constituyen la guía para el ciclo de vida de la ingeniería de aplicación.

Todos estos productos son resultado de la **gestión global del dominio** y de las actividades específicas de **análisis, diseño e implementación del dominio,** que se describen en los cuatro primeros apartados subsiguientes. El último apartado describe la **construcción del repositorio,** que es un nexo de unión entre los dos procesos de esta metodología:

- *La ingeniería de dominio almacena todos los elementos generados durante las actividades de análisis, diseño, implementación y gestión del dominio en el repositorio. Este proceso es el responsable de la construcción y gestión de dicho repositorio.*
- *La ingeniería de aplicación accede al repositorio para seleccionar, usar y o adaptar los elementos que requiera para la construcción de nuevos sistemas. Las adaptaciones y elementos de nueva creación también serán almacenados en el repositorio.*

3.2.1 Análisis del dominio.

Las principales tareas realizadas en esta etapa son:

- **Definición del dominio.**
- **Especificación del mismo.**
- **Validación de esta especificación.**
- **Soporte a la ingeniería de aplicación.**

La ingeniería inversa ha tenido un papel relevante en el desarrollo de estas actividades. En general, en el contexto de la ingeniería de dominio, la ingeniería inversa constituye una estrategia sistemática para obtener información de los proyectos existentes en el dominio. En particular, en el presente trabajo, el estudio y análisis de la información recopilada de este modo ha permitido obtener:

- *Mayor comprensión del problema,* lo cual ha contribuido positivamente en la definición del dominio.
- *Mayor conocimiento de las soluciones adoptadas en el pasado,* lo cual ha constituido una importante fuente de información para el diseño del dominio.

3.2.1.1 Definición del dominio.

Una correcta definición del dominio requiere: *delimitar el ámbito del dominio, describir el mismo, su operativa y la relación con su entorno, identificar las similitudes y variaciones de la familia de productos, determinar la viabilidad técnica y económica del dominio, y establecer un glosario de términos y acrónimos.*

3.2.1.1.1 Ámbito del dominio.

El ámbito del dominio está restringido al software de control para FMSs aplicados al mecanizado por arranque de viruta, enmarcados en el área de la fabricación discreta y preferiblemente integrados en un entorno CIM.

En este contexto, un FMS consta de un grupo de FMCs y/o máquinas CNC conectadas a través de un sistema de manipulación automatizado encargado del flujo de materiales (tanto de entrada como de salida), interconectadas mediante una red de comunicaciones y controladas a través de uno o varios ordenadores.

3.2.1.1.2 Descripción y operativa del dominio.

El dominio del software de control para FMSs está constituido por la familia de productos a cargo de la planificación, el control y el seguimiento de todas las actividades desempeñadas en la planta de fabricación, así como la integración del propio sistema de control con las restantes funciones productivas de la empresa.

El principal objetivo de todo sistema de control de estas características es conseguir que se fabriquen un conjunto de piezas de acuerdo al plan de producción vigente. Para ello, el sistema de control debe establecer la secuencia de operaciones de producción, supervisar su correcta ejecución, coordinar todos los recursos disponibles, y gestionar los flujos de materiales, herramientas e información. No obstante, su labor se complica debido a las frecuentes variaciones en los pedidos, cambios en los objetivos de producción, así como averías de máquinas o problemas de falta de materiales.

Dada la complejidad de los sistemas en estudio, puede resultar materialmente imposible definir la **operativa del dominio** considerando el proceso en su conjunto. Por ello, *los flujos de piezas, de herramientas y demás acciones se descomponen en tareas unitarias:*

- **Ejecución de un plan.** Conjunto de operaciones necesarias para ejecutar un plan de trabajo (con o sin replanificación).
- **Tareas relacionadas con la manipulación de piezas:**
  - *Entrada de piezas a la planta de fabricación.* Existen cuatro tareas alternativas: paletizar la pieza de forma manual, paletizar la pieza de forma automática, no paletizar la pieza e introducirla manualmente (un operario) o no paletizar la pieza e introducirla automáticamente (un robot).

En los dos primeros casos, el paletizado incluye: conseguir un palet libre, colocar los utillajes, depositar la pieza y poner el palet en la posición de entrada a la planta. Cuando el paletizado es automático probablemente se necesite el cambio de garras del robot.

En los dos últimos casos, la pieza procedente del almacén central de materia prima se deposita en un almacén intermedio de piezas o directamente en máquina.

- *Transporte de piezas en la planta de fabricación.* El palet se traslada desde una posición origen a una posición destino. Cuando el palet entra en la planta, el destino puede ser una posición de carga/descarga a máquina o un buffer de almacenamiento intermedio de espera a la asignación de posición de carga/descarga. Cuando el palet sale de la planta, el destino es la posición de salida designada.
- *Carga descarga de piezas en de la máquina.* Hay tres alternativas: cargar la pieza desde un palet (o con el palet) situado en la posición de carga/descarga de la máquina, cargar la pieza desde un almacén intermedio de piezas o cargar la pieza del almacén central de materia prima.

  Cada una de estas tareas tiene la correspondiente tarea de descarga de piezas, pero en la tercera opción, la descarga es en el almacén de producto acabado, no de materia prima. La ejecución de alguna de estas tareas puede requerir el cambio de garras del robot.
- *Salida de piezas de la planta de fabricación.* Se distinguen las siguientes tareas alternativas correspondientes a las tareas alternativas de entrada de piezas: salida de palet con despaletizado manual, salida de palet con despaletizado automático, salida de pieza de forma manual o salida de pieza de forma automática.

- **Tareas relacionadas con la manipulación de herramientas:**
  - *Reserva de herramientas para la ejecución de un plan de trabajo.*
  - *Las tareas de entrada, transporte, carga descarga y salida de herramientas* se corresponden con las descritas para las piezas, sustituyendo el almacén

intermedio de piezas y el almacén central de materia prima por los correspondientes a herramientas.

- **Tareas relacionadas con la manipulación de programas:**
  - *Modificación de los programas de control numérico* con las herramientas que se han cargado en el carrusel de una máquina para realizar un mecanizado concreto.
  - *Carga inicial de programas.* Se cargan en las máquinas de control numérico y en los robots aquellos programas de uso general, los utilizados con mayor frecuencia y los que previsiblemente se van a necesitar en breve, aprovechando al máximo la capacidad de memoria del dispositivo.
  - *Carga posterior de programas.* Los programas necesarios que no estén en los dispositivos correspondientes se cargan en sustitución de otros ya empleados, utilizando tiempos muertos.
  - *Carga de correctores de herramientas.* Finalizada la carga de programas de control numérico en máquina, se cargan los correctores de herramientas.
  - *Ejecución de programas.* La ejecución de los programas de robot y de los programas de mecanizado de atadas[21].

- **Tareas de mantenimiento.** Además de las tareas manuales identificadas anteriormente, el operario debe realizar las siguientes:
  - *Reposición de materia prima en el almacén central.*
  - *Reposición de materia prima en el almacén intermedio.*
  - *Revisión de dispositivos.*
  - *Revisión de las posiciones del transporte.*

- **Tareas generales:**
  - *Solicitud de estado de un dispositivo concreto.*
  - *Solicitud de fin de proceso de un módulo.*

---

[21] Por atada se entiende el conjunto de operaciones a realizar sobre la pieza sin cambiar el amarre.

3.2.1.1.3 Relación del dominio con su entorno.

De forma genérica, la información que fluye hacia el sistema de control determina qué y cuándo se debe fabricar (MRP/Planificador off-line) y cómo fabricar cada producto (CAD/CAM/CAPP). Además, tanto la planificación como el diseño reciben las indicaciones pertinentes desde los sistemas de mantenimiento off-line y control de calidad.

Toda esta información (qué, cómo y cuándo fabricar) llega a los dispositivos de planta a través del sistema de control, que actúa como *intermediario entre la planta de fabricación y los sistemas software de nivel superior*. Estas relaciones se recogen en la figura 3.2, que es una modificación del diseño en Y del profesor Scheer *[Scheer, 91]*.

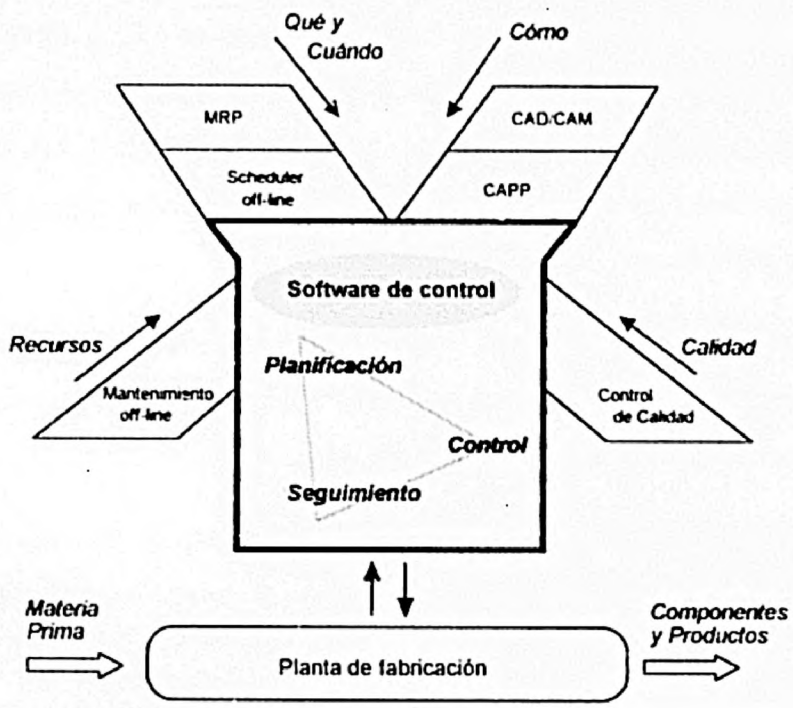


Figura 3.2: Relación entre el software de control para FMSs y su entorno.

3.2.1.1.4 Variabilidad en el dominio.

A primera vista podría parecer que estos sistemas tienen grandes similitudes. Sin embargo, las variaciones son muchas de una instalación a otra: las máquinas y demás recursos varían tanto en número como en tipo, la disposición en planta (layout), el tipo

de pedidos, el conjunto de piezas a fabricar, los procesos de mecanizado, la manutención y transporte de materiales y herramientas, y la lógica de control y decisión.

3.2.1.1.5 Viabilidad del dominio.

Pero precisamente esta variabilidad y complejidad, características inherentes a los problemas de este dominio, justifican el estudio de la viabilidad de aplicar la megaprogramación a esta familia de productos, para construir una solución genérica que pueda ser adaptada a las distintas aplicaciones de control.

La decisión de realizar este estudio se justifica **por tratarse de un dominio:**

- **Medianamente maduro.** Se han realizado muchos proyectos sobre el tema, se han diseñado diferentes arquitecturas genéricas y modelos de referencia, existen numerosas instalaciones en el ámbito industrial, pero en la actualidad se sigue investigando la aplicación de nuevas metodologías y tecnologías al desarrollo de este tipo de software.
- **Razonablemente estable.** Los requisitos del sistema no presentan cambios drásticos continuos, aunque sea necesario añadir nuevas funcionalidades como consecuencia de la evolución natural del dominio.
- **Económicamente viable.** La demanda actual y futura de este tipo de sistemas justifica el esfuerzo y la inversión que conlleva un programa de reutilización.

3.2.1.1.6 Glosario de términos y acrónimos.

La definición del dominio presenta gran cantidad de acrónimos y conceptos. Por ello, *se considera indispensable generar glosarios de términos y acrónimos* (Apéndices A y B respectivamente), que permitan establecer una vía de comunicación efectiva entre las partes involucradas. En realidad, son elementos esenciales de esta etapa, objeto de ampliación en fases posteriores.

3.2.1.2 Especificación del dominio.

En la especificación del dominio *se definen los requisitos funcionales y no funcionales que debe cumplir el software de control para FMSs, y la exigencia de que éste sea reutilizable.*

Del estudio teórico/práctico realizado, analizando el estado del arte del desarrollo de proyectos de sistemas de control para FMSs y las tendencias de la fabricación, los requisitos funcionales se han subdividido en funcionalidades actuales y futuras, y se ha exigido además que los sistemas de control deben ser altamente reutilizables.

3.2.1.2.1 Requisitos funcionales.

En las tablas 3.1 y 3.2 se han agrupado los requisitos funcionales en requisitos actuales y futuros de los sistemas de control para FMSs.

Requisitos funcionales actuales

Requisito	*Descripción*
Planificación de las operaciones de planta	Generación de planes con la secuencia óptima de operaciones para cada una de las máquinas. Además, el sistema debe tener capacidad para *reaccionar ante imprevistos y adaptar el plan generado off-line a la situación actual* de la planta de fabricación.
Coordinación general de las operaciones de planta	Control del flujo de materiales y de información a través del sistema físico de producción, con objeto de cumplir el plan de trabajo. *Si se producen desviaciones relevantes respecto al plan, será preciso generar una nueva secuencia de operaciones.*

Tabla 3.1: Requisitos funcionales actuales.

Requisito	*Descripción*
Gestión de piezas y palets	El sistema debe gestionar el traslado de los palets desde el buffer central o local junto a las máquinas hasta la zona de paletizado, comprobar la disponibilidad de las piezas y palets, y ordenar el amarre de las piezas en los palets antes y después de las operaciones de mecanizado.
Gestión de programas	Mantenimiento de la base de datos con los programas de mecanizado de las piezas, y descarga de dichos programas en las máquinas de control numérico cuando el programa correspondiente no esté almacenado en la memoria local de la máquina en cuestión.
Gestión de herramientas	Mantenimiento de la base de datos con toda la información actualizada referente a: características, dimensiones reales y nominales, y vida remanente. Además, el sistema de control debe suministrar las herramientas necesarias y sus dimensiones reales para que el CNC de la máquina pueda realizar las correcciones pertinentes sobre las dimensiones nominales que asumen los programas de mecanizado. También ordena a los operarios que preparen nuevas herramientas cuando se exceda la capacidad del sistema, y que se trasladen al almacén central aquellas que no sean necesarias. En caso de ruptura de una herramienta, debe tratar de encontrar otra que la reemplace en el almacén local, y si no la encuentra, recurrir al almacén central.

Tabla 3.1 (continuación).

Requisito	*Descripción*
Gestión del transporte	Control y supervisión del movimiento de las piezas paletizadas y de las herramientas entre estaciones de trabajo dentro del FMS. En el caso de trabajar con varios AGVs, se precisa un módulo específico para evitar colisiones, que actúe en función de la información procedente de los sensores.
Gestión de la información	Los diversos módulos que componen el sistema de control deben trabajar coordinadamente, siendo de vital importancia que dispongan de *información actualizada en todo momento.*
Monitorización	Visualización en todo momento del estado actual de los instrumentos, variables de proceso, maquinaria y del propio producto, de modo que se pueda obtener una producción libre de errores.
Interface de usuario	Comunicación con el operario, con todo tipo de ayudas y facilidades para simplificar su manejo.

Tabla 3.1 (continuación).

Requisitos funcionales futuros

Requisito	*Descripción*
Mantenimiento de los equipos	Generación de esquemas de mantenimiento preventivo, predictivo y correctivo, guardando un histórico de las operaciones realizadas en cada equipo.

Tabla 3.2: Requisitos funcionales futuros.

Requisito	*Descripción*
Gestión de errores y de situaciones anómalas	Las anomalías (situaciones anormales contempladas por los procedimientos) deben ser previstas y tenidas en cuenta a la hora de realizar el diseño del control, previniendo las respuestas apropiadas. Los errores o fallos del sistema (anomalías graves sin acción correctora asociada) requieren procedimientos de recuperación de errores, que permitan continuar trabajando y minimicen los daños ocasionados en el sistema.
Control del proceso estadístico	Comprobación periódica de las desviaciones existentes para tomar las medidas oportunas, que permitan corregir cualquier variación en el proceso de fabricación.

Tabla 3.2 (continuación).

3.2.1.2.2 Requisitos no funcionales.

A los requisitos funcionales anteriores se añaden los no funcionales. Éstos son un conjunto de *requisitos de interface, operación, calidad, y desarrollo* que debe cumplir el software de control para FMSs.

Requisito	*Descripción*
Flexible	La flexibilidad exigida a los FMSs no podrá obtenerse si ésta no reside también en *el sistema de control*, que *es el responsable de explotar adecuadamente la capacidad potencial del sistema.*

Tabla 3.3: Requisitos no funcionales.

Requisito	*Descripción*
Abierto	El sistema de control debe disponer de procedimientos de comunicación estándares, que permitan el intercambio de mensajes entre aplicaciones en un entorno heterogéneo, sin depender del hardware o sistema operativo. Por lo tanto, *el software de control debe ser portable, interoperable, adaptable, y debe estar integrado con las otras funciones productivas.*
Modular	Un diseño modular facilita las modificaciones en la funcionalidad del sistema (una simple adición o supresión de módulos) y *fomenta la reutilización*, siempre y cuando los módulos desempeñen una funcionalidad específica.
Configurable	Las variaciones que sufre un FMS a lo largo de su vida útil (disponibilidad de recursos, cambios en sus parámetros, reemplazamiento o mejora de ciertos elementos, etc.) precisan un alto grado de reconfiguración del software de control. Este aspecto también es de gran utilidad en caso de funcionamiento en modo degradado.
Operativo en tiempo real	El flujo de información debe ser rápido y adecuado a las necesidades de los módulos del sistema de control, para articular los mecanismos de reacción precisos. Así, tareas de generación de informes, monitorización, etc. conceden tiempos de respuesta mayores, que el software del nivel de dispositivo que debe responder a los eventos recibidos de los controladores de planta.

Tabla 3.3 (continuación).

Requisito	*Descripción*
Sencillo de concepto y manejo	Una arquitectura de control sencilla ayuda a comprender la funcionalidad de sus componentes, aumentando la confianza y la credibilidad del personal, que debe disponer de la información y ayudas necesarias para todas aquellas operaciones en las que se vea involucrado.
Fiable, robusto y tolerante a fallos	Como elemento integrador, el sistema de control debe tener un alto grado de fiabilidad, ya que su fallo puede llegar a colapsar todas las operaciones en marcha. Más aún, *debe ser capaz de recuperarse automáticamente, o en su defecto, funcionar en modo degradado.*
Eficiente en la utilización de los recursos	La fuerte inversión que supone la adquisición de los recursos exige que el sistema de control optimice la asignación y empleo de los mismos, de acuerdo con planes de trabajo realistas y viables en la práctica.
Expandible	La posible ampliación del sistema físico debe ir acompañada de la correspondiente del software de control para añadir las nuevas funcionalidades; este proceso debe ser sencillo.
Costo adecuado	La construcción del software de control es una tarea compleja, que consume muchos recursos y que exige un nivel de calidad adecuado al grado de rendimiento que se quiere obtener. El costo del hardware y el software de control debe estar ajustado al costo global del sistema productivo.

Tabla 3.3 (continuación).

3.2.1.3 Validación de la especificación del dominio.

La *validación de la definición del dominio y la de su especificación* se realizan de la siguiente forma.

Validación de la definición del dominio

La definición del dominio se ha validado en relación con la bibliografía especializada, la experiencia adquirida con la participación en la construcción de distintas células y sistemas de fabricación flexibles en el marco del Master en Robótica y Automatización, del Master en Tecnologías Avanzadas de Fabricación, y de proyectos CICYT *[CICYT, 94], [CICYT, 95], los contactos mantenidos con especialistas de prestigio internacional*[22]*, y las necesidades reales de las PYMES manufactureras.*

Validación de la especificación del dominio

La especificación se ha validado mediante un proceso manual de revisión de los requisitos, teniendo especial cuidado en no confundir: los requisitos funcionales y no funcionales, los objetivos del sistema, y la información de diseño. Esta confusión se presenta habitualmente cuando se emplea el lenguaje natural como herramienta de especificación.

Mediante este proceso se ha comprobado que los requisitos, y especialmente **los requisitos funcionales, son: completos** (especifican todos los servicios), **y consistentes** (no son contradictorios).

Esta revisión manual exhaustiva es la única posibilidad para comprobar que los requisitos son completos. La consistencia también se puede comprobar automáticamente usando alguna herramienta, cuando los requisitos se expresan en lenguaje formal.

---

[22] Robert Maglica y Martin Fabian de la Universidad de Chalmers (Suecia), Sushil Birla de la Universidad de Michigan (USA) o Jeffrey S. Smith de la Universidad de Texas (USA).

Por otra parte, no se ha considerado preciso usar un simulador en esta etapa, ya que el coste y el tiempo asociados no compensan, sobre todo teniendo en cuenta que se trata de un dominio razonablemente estable.

La validación de la especificación del dominio es una tarea altamente crítica. Cuando es inadecuada se propagan al diseño e implementación del sistema los errores producidos en los requisitos, resultando costoso corregirlos a posteriori.

También cabe destacar que la definición y la especificación del dominio son auto-consistentes y, además, la especificación es consistente respecto a la definición. Esta última propiedad significa que la especificación establece los requisitos funcionales y no funcionales de los sistemas potenciales que se ajustan a las características impuestas en la descripción de tales sistemas.

3.2.1.4 Soporte a la ingeniería de aplicación.

Para generar la especificación de una instancia de la familia de productos se debe definir el conjunto de requisitos y decisiones que un ingeniero de aplicación debe resolver.

A tal efecto *se establecen los procesos para obtener la operativa del nuevo sistema y sus requisitos funcionales*, que se presentan a continuación.

Proceso para establecer la operativa del nuevo sistema

La operativa general del nuevo sistema se describe mediante el conjunto de tareas que se van a realizar sobre la planta concreta. Estas tareas son *un subconjunto de las tareas identificadas para el dominio*, e indican cómo se va a llevar a cabo la manipulación de piezas, de herramientas y de programas, las operaciones de mantenimiento y demás acciones. Toda esta información *se organiza según se establece en la tabla 3.4.*

Información	*Detalle*
Almacenes	Se describen los almacenes generales y locales de materia prima, herramientas, palets, utillajes, garras, mordazas, etc., indicando si son automáticos o manuales.
Piezas	Por cada familia de piezas se presenta el identificador de la familia, la descripción, los identificadores de las piezas, y las posibles rutas de fabricación. Para cada ruta de fabricación se describe el modo de paletizado, el modo de entrada al sistema y el modo de salida del sistema correspondientes a la pieza.
Herramientas	Se especifica la base de datos de herramientas y si es necesario comprobar la disponibilidad de las herramientas antes de la ejecución del plan. Para cada uno de los posibles destinos de las herramientas se precisan los datos sobre el modo de paletizado, el modo de entrada al sistema y el modo de salida del sistema correspondientes a las herramientas.
Operaciones de paletizado	Tanto en el caso de paletizado manual como automático se indica el elemento objeto del paletizado, el tipo de palet, el tipo de utillaje, el tipo de garra y la descripción de la operación.
Máquinas	Por cada dispositivo de mecanizado se adjunta el identificador, el tipo de máquina, la descripción, el protocolo de comunicaciones, la capacidad del carrusel, el modo de carga/descarga de herramientas, el modo de carga/descarga de programas de control numérico, y el modo de carga de los correctores de herramientas. Para cada operación de mecanizado se indica la pieza a mecanizar, la mordaza necesaria, el modo de carga/descarga de la pieza y la descripción de la operación.

Tabla 3.4: Información sobre la operativa del nuevo sistema.

Información	*Detalle*
Sistemas de manipulación	Por cada uno de ellos se precisa el identificador, el tipo de sistema, la descripción, el protocolo de comunicaciones, el modo de carga/descarga de programas y las posiciones de acceso. Para cada operación de manipulación se especifica el elemento que manipula (pieza, herramienta, utillaje, etc.), los elementos de sujeción requeridos y la descripción de la operación.
Operaciones de mantenimiento	Para cada una de ellas se establece el elemento objeto del mantenimiento, el responsable de llevarlo a cabo y la descripción de la operación.

Tabla 3.4 (continuación).

Proceso para establecer los requisitos funcionales

Partiendo de los datos de modelado de la planta de fabricación (modelos de pedidos, de piezas, de procesos, de operaciones y de recursos), de las especificaciones técnicas de los distintos componentes del sistema, y del layout (disposición en planta de los elementos), el ingeniero de aplicación debe *responder a las preguntas* que se indican en la tabla 3.5.

Conclusión

La especificación de la instancia de la familia de productos se obtiene particularizando la especificación del dominio. *La aprobación de la especificación generada de este modo es responsabilidad del cliente.*

¿Se desea realizar			
adaptación del plan de producción a la situación real de la planta?	❒ No	❒ Si	❒ .....
coordinación general de los flujos?	❒ No	❒ Si	❒ .....
gestión de piezas?	❒ No	❒ Si	❒ .....
gestión de herramientas?	❒ No	❒ Si	❒ .....
gestión de palets?	❒ No	❒ Si	❒ .....
gestión de programas?	❒ NC	❒ RC	❒ .....
gestión de los sistemas de transporte?	❒ No	❒ Robot	❒ .....
monitorización del sistema?	❒ No	❒ Si	❒ .....
modo de funcionamiento?	❒ Automático		❒ Semi-Aut.
operaciones de mantenimiento?	❒ No	❒ Si	❒ .....
tareas de diagnóstico?	❒ No	❒ Si	❒ .....
control del proceso estadístico?	❒ No	❒ Si	❒ .....
Observaciones:			

Tabla 3.5: Requisitos funcionales de un nuevo sistema.

3.2.2 Diseño del dominio.

Las principales tareas que se realizan en esta etapa son:

- **Diseño o selección de una arquitectura software genérica para el dominio, identificación de sus módulos y de las relaciones entre los mismos, especificación de su mensajería.**
- **Diseño de las arquitecturas de los distintos módulos de la arquitectura genérica del dominio.**
- **Verificación de estas arquitecturas y de la del dominio.**
- **Desarrollo del soporte a la ingeniería de aplicación.**

A continuación se presentan y verifican todas estas arquitecturas, así como el soporte a la ingeniería de aplicación.

3.2.2.1 Arquitectura genérica del dominio propuesta.

3.2.2.1.1 Selección de la arquitectura.

Considerando la condición de reutilización del software que hemos impuesto a los sistemas de control para los FMSs, la arquitectura genérica del dominio debe soportar la reutilización de diseño y de código del software de control de los sistemas dinámicos de eventos discretos. Por lo que esta arquitectura debe ser *de tipo distribuido y combinar las características de los sistemas orientados a objetos y de los sistemas de eventos.*

De acuerdo con la evaluación del impacto de las arquitecturas actuales en la reutilización, estos aspectos están contemplados en gran parte en la arquitectura genérica propuesta por Robotiker *[Robotiker, 92]*, en la cual los módulos están relacionados jerárquicamente siguiendo un esquema cliente/servidor.

La arquitectura propuesta en la tesis está inspirada en la de Robotiker, sustituyendo las relaciones jerárquicas por relaciones distribuidas en las que cualquier módulo puede

ser cliente o servidor en un momento dado (figura 3.3), lo cual le dota de mayor flexibilidad, disminuye la complejidad en la construcción de los módulos, e incrementa el conjunto de posibles configuraciones para los sistemas del dominio.

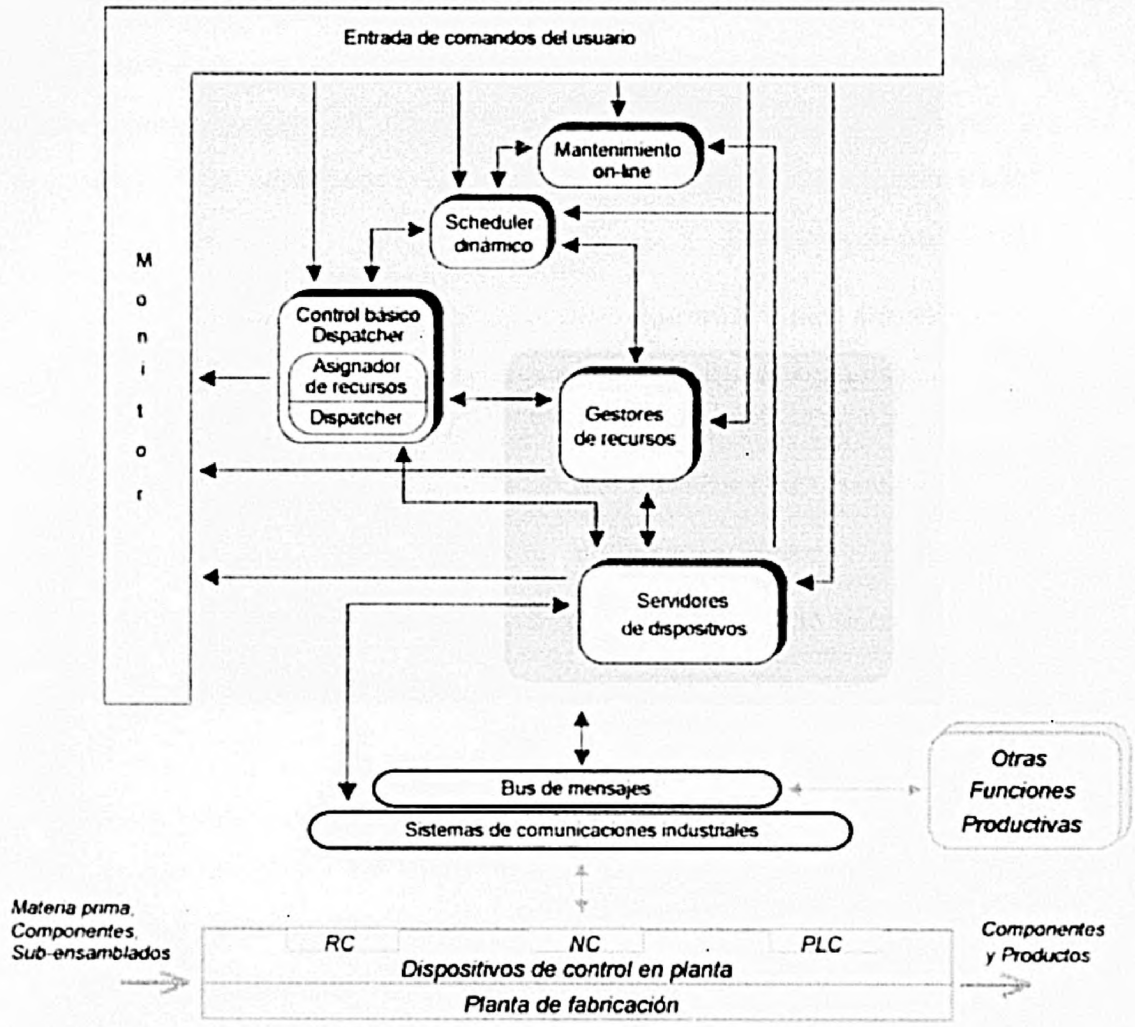


Figura 3.3: Arquitectura genérica del dominio.

3.2.2.1.2 Identificación de los principales módulos.

Tal y como se muestra en la figura 3.3, la arquitectura genérica del dominio está compuesta de un conjunto de módulos relacionados de forma distribuida a través de peticiones de servicio y eventos, cuyas tareas específicas se definen a continuación.

- **Monitor.** Realiza la *visualización del proceso en planta,* proporciona la *interface con el usuario,* permite el *cambio del modo de funcionamiento* del sistema (automático o semi-automático) y *envía comandos* a los controladores de dispositivo cuando funciona en modo semi-automático.

- **Mantenimiento (on-line).** Se encarga de las *tareas de diagnóstico y tratamiento* de los recursos, con objeto de minimizar los tiempos de inactividad no previstos. Aplica mantenimiento predictivo, integrado con el preventivo, el correctivo y el diagnóstico de máquinas.

- **Scheduler dinámico (on-line).** *Adapta la estrategia de funcionamiento y la secuencia de operaciones,* de acuerdo con la información recibida sobre el plan de producción generado, el estado de la planta y las alteraciones en los pedidos. Complementa al Scheduler off-line o sistema de planificación a capacidad finita.

- **Control básico Dispatcher (CBD).** *Ejecuta y coordina las actividades de planta* requeridas para fabricar todas las piezas contenidas en el plan de producción. Está constituido por los módulos:

  - *Asignador de recursos.* Asigna los recursos para llevar a cabo las distintas operaciones, lo cual tiene sentido cuando no existe un Scheduler, o existiendo, únicamente establece la secuencia de piezas a fabricar.
  - *Dispatcher.* Encadena, coordina y sincroniza las operaciones ya asignadas a recursos según el plan de producción establecido.

- **Gestores de recursos.** Desempeñan *tareas especializadas en el tratamiento de recursos de difícil gestión,* facilitando y simplificando de este modo la labor del módulo CBD. Posibles gestores son:

  - *Gestor de herramientas.* Gestiona este tipo de recurso, controla su ubicación y mantiene la base de datos con información actualizada de las mismas.
  - *Gestor DNC.* Gestiona los programas de control numérico y los adapta con las herramientas concretas que se empleen en el mecanizado.

- *Gestor de transporte.* Controla la situación actual de piezas y/o herramientas, los caminos óptimos, las posibles colisiones, etc. Este módulo es necesario cuando la complejidad de gestión excede el ámbito del servidor de dispositivo correspondiente.

- **Servidores de dispositivos.** Ofrecen una *imagen homogénea del comportamiento de los dispositivos físicos* y resuelven aspectos tales como: la conexión física con los controladores de dispositivos, sus protocolos de comunicación, así como otras particularidades específicas de dichos equipos.

- **Bus de mensajes.** Permiten la *comunicación y sincronización* entre los módulos distribuidos que componen el sistema.

- **Sistemas de comunicaciones industriales.** Establecen la *conexión* entre módulos y con los dispositivos de planta.

3.2.2.1.3 Relación entre los módulos de la arquitectura.

Los módulos identificados se relacionan **a través de peticiones de servicio y eventos:**

- *Las peticiones de servicio son mensajes que solicitan la ejecución de una acción determinada o el envío de cierta información.*

- *Los eventos son mensajes que notifican la finalización de un servicio o una alteración en el estado de la planta.*

Esto significa que *la operativa del sistema está marcada por el flujo de mensajes* entre los diversos módulos, y que *su comportamiento dinámico está definido por los modelos de eventos y servicios*, que representan cómo se realiza el intercambio de mensajes entre módulos.

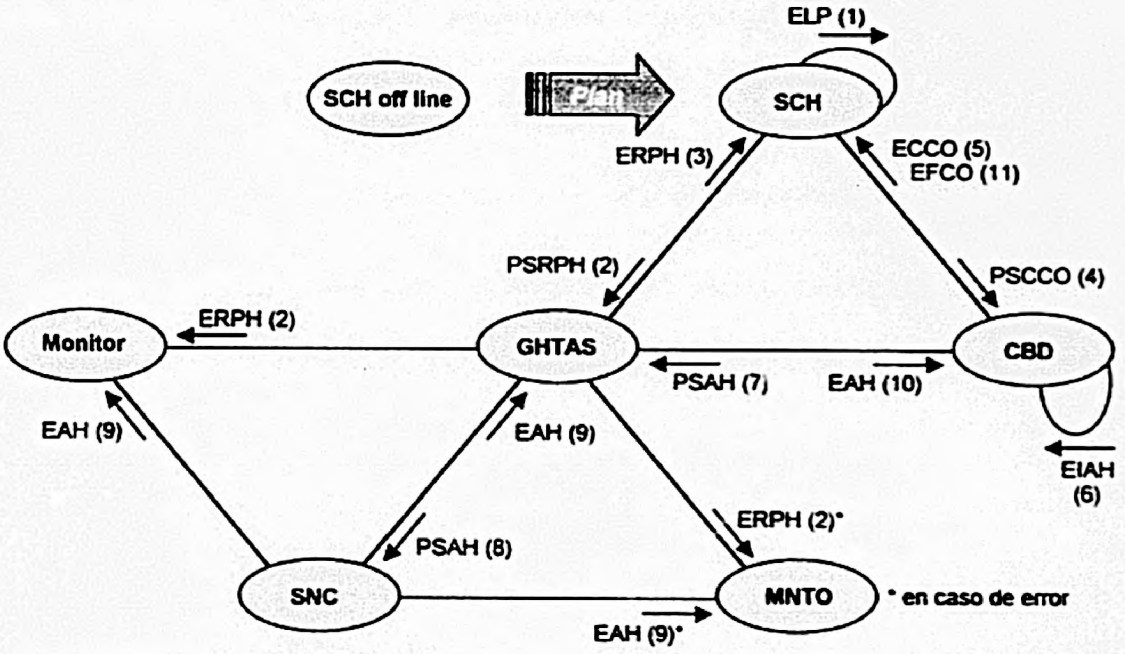
Así pues *se completa la definición de la arquitectura generando los diagramas de flujo de mensajes (MFD - Message Flow Diagram) y el documento de especificación de servicios y eventos del sistema global.* Este documento registra la definición detallada de los mensajes según los modelos de eventos y de servicios.

Diagramas de flujo de mensajes

Los MFDs identifican los objetos y representan el intercambio de mensajes entre ellos, determinando de este modo el comportamiento dinámico del sistema.

Para cada una de las tareas identificadas en la definición del dominio se realiza: un MFD, la definición detallada de los servicios y eventos emitidos por los módulos involucrados, y el orden en que se generan dichos mensajes.

A modo de ilustración, se muestra en la figura 3.4 el MFD para la tarea de ejecución de un plan de trabajo. Las elipses representan los módulos, las flechas identifican el intercambio de mensajes, y los números establecen la secuencia de dicho intercambio.



1 Auto-evento de Lanzamiento de un Plan
2 Petición de Servicio de Reserva Previa de Herramientas
3 Evento de Reserva Previa de Herramientas
4 Petición de Servicio de Comienzo del Conjunto de Operaciones
5 Evento de Comienzo del Conjunto de Operaciones
6 Auto-evento de Inicio de Actualización de Herramientas
7 Petición de Servicio de Actualización de Herramientas
8 Petición de Servicio de Actualización de Herramientas
9 Evento de Actualización de Herramientas
10 Evento de Actualización de Herramientas
11 Evento de Finalización del Conjunto de Operaciones

Figura 3.4: MFD para la ejecución de un plan de trabajo.

En la elaboración de la lista de servicios y eventos se tienen en cuenta las siguientes recomendaciones:

- A cada una de las posibles acciones que puedan ser solicitadas a un módulo se asigna un servicio.
- A cada una de las posibles peticiones de información que puedan ser solicitadas a un módulo se asigna un servicio.
- Cada módulo dispone de servicios de inicialización y configuración.
- Resulta conveniente asociar un evento con la finalización de un servicio. Esta asociación es imprescindible cuando el final de la ejecución de un servicio es condición para el comienzo de otra acción.
- Los valores de las variables de estado de la planta están asociados normalmente a eventos, de forma que la alteración de estos valores sea notificada a los módulos interesados.
- Un determinado evento puede llevar asociado una serie de servicios adicionales de inicialización, activación y desactivación.

Modelo de eventos y modelo de servicios

La definición detallada de los servicios y eventos asociados con cada MFD se realiza con independencia del módulo que los implementa; es decir, *el modelo de servicios (tabla 3.6) y el modelo de eventos (tabla 3.7)* definidos consideran el módulo como un parámetro asociado a los mismos.

Estos modelos *recogen la información necesaria para la perfecta definición de su interface, abstrayéndose del lenguaje de programación que se emplee para su implementación.*

Cabecera del servicio	
Módulo:	Nombre del módulo que implementa el servicio
Servicio:	Nombre del servicio
Id_servicio:	Identificador del servicio (único)
Interface:	Vía de comunicación
Versión:	Número de versión
Clientes:	Solicitantes potenciales del servicio
Descripción:	Tarea que realiza el servicio
Cuerpo del servicio	
Sincrono/Asincr [Aceptación, Fin servicio, Evento asociado] *{[retorno] NombreServicio (parámetros)}*	

Tabla 3.6: Modelo de servicios.

Cabecera del evento	
Módulo:	Nombre del módulo que genera el evento
Evento:	Nombre del evento
Id_evento:	Identificador del evento (único)
Versión:	Número de versión
Condición:	Condición que desencadena el disparo del evento
Cuerpo del evento	
{[retorno] NombreEvento (parámetros)}	
Lista de receptores del evento	
Módulo:	Nombre del módulo que recibirá el evento
Interface:	Vía de comunicación
Comentario	

Tabla 3.7: Modelo de eventos.

3.2.2.1.4 Especificación de mensajería de los módulos.

A partir de los MFDs y la información asociada a los mismos *se genera el documento de especificación de servicios y eventos del sistema global*, el cual constituye un elemento clave para el desarrollo de los componentes. En dicho documento se recoge *la especificación de mensajería de cada módulo, que se obtiene extrayendo los servicios y eventos que debe implementar, los servicios que puede solicitar y los eventos que puede recibir de otros módulos.*

3.2.2.2 Arquitecturas genéricas de los módulos del dominio.

A continuación *se analiza la funcionalidad de los distintos módulos y se propone una arquitectura para cada uno de ellos que incorpora la reutilización en la línea que persigue la metodología con reutilización sistemática.* Estas arquitecturas están inspiradas en trabajos de tesis anteriores, *[Alvarez, 95], [Burgos, 97], [López, 95]*, en los cuales los autores proponen determinadas arquitecturas específicas.

3.2.2.2.1 Módulo Monitor.

Funcionalidad

El módulo Monitor proporciona una *representación gráfica actualizada del estado del FMS*, que ofrece en todo momento una imagen simplificada de la planta con la máxima información sobre las actividades que tienen lugar en la misma, *incluidas las alarmas.*

La actualización de esta imagen se obtiene mediante el tratamiento de los mensajes recibidos (en su mayoría eventos) de los restantes módulos del sistema de control.

La *interface de usuario* dispone de ayuda on-line, información de las distintas opciones y acceso a dichas opciones a través del teclado y del ratón. De las posibles peticiones de usuario caben destacar las siguientes: cambio de modo de funcionamiento (automático

o semi-automático), arranque del sistema, parada total del sistema, y pausa/reanudación del sistema global o de un área específica.

También se mantiene un *registro histórico* de los mensajes recibidos y enviados, con acceso al mismo para obtener una descripción breve y general sobre la evolución de las actividades, o el detalle de los mensajes relacionados con cierto dispositivo.

Arquitectura

La arquitectura genérica del módulo Monitor se presenta en la figura 3.5 y la funcionalidad de sus componentes se describe brevemente a continuación:

- **Configurador.** Establece las condiciones iniciales de acuerdo con la *configuración de la sesión.* La configuración se realiza de forma interactiva, guardando copia de las configuraciones usadas con anterioridad.
- **Gestor de peticiones de usuario.** Establece la *comunicación con el usuario:* recibe peticiones de solicitud de ayuda, cambio de funcionalidad o información de eventos, y presenta las correspondientes respuestas.
- **Ayuda.** Determina el tipo de ayuda y selecciona los ficheros que la contienen.
- **Gestor de servicios y eventos.** *Procesa las peticiones de servicio y los eventos* que van a ser enviados, así como los mensajes que se reciban de otros módulos.
- **Registro.** *Almacena todo mensaje recibido o enviado* por este módulo en el fichero de incidencias.
- **Actualizador.** *Modifica los datos de estado* (forma, color y posición) de los objetos relacionados con el último evento procesado, de modo que se mantenga la coherencia con el estado real de la planta.
- **Representación gráfica.** *Actualiza la representación gráfica* de la planta, reflejando las nuevas formas, colores y posiciones de objetos que lo requieran.
- **Interface con otros módulos.** Responsable del *intercambio de mensajes* con los restantes módulos del sistema de control.

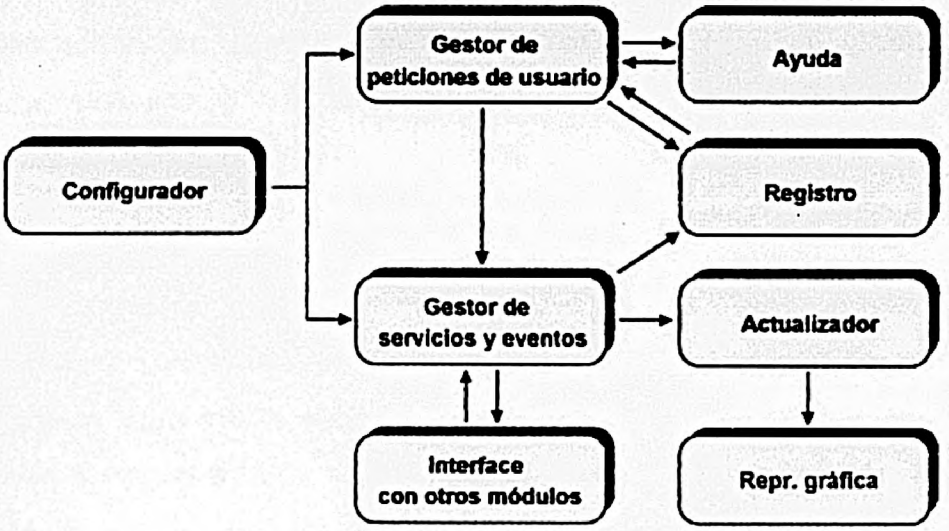


Figura 3.5: Arquitectura genérica del módulo Monitor.

3.2.2.2.2 Módulo Mantenimiento (on-line).

Funcionalidad

Mantenimiento off-line genera un plan de mantenimiento para un determinado periodo de tiempo teniendo en cuenta los recursos necesarios para llevarlo a cabo. Para ello dispone de información sobre los mantenimientos planificados de los distintos dispositivos de planta (tipo y frecuencia), y sobre los elementos que precisan una monitorización continua de condiciones.

También analiza los bancos de históricos generados por el FMS y los datos de fabricante de los equipos, y aplica criterios de coste de las acciones de mantenimiento. Con ello obtiene nuevos tiempos de mantenimiento preventivo para los equipos, y sugerencias de eliminación de acciones poco eficaces, de inclusión de ciertos equipos en el mantenimiento preventivo o de cambio de alguna estrategia.

El plan óptimo resultante se envía a Mantenimiento on-line, que se encarga de gestionar su ejecución de acuerdo con producción, y de realizar aquellas modificaciones que considere oportunas basándose en la situación real de las operaciones de mantenimiento.

Mantenimiento on-line también recibe datos de la planta, vía los servidores de dispositivos y los sistemas de adquisición de datos. A partir de esta información, este módulo establece un *diagnóstico del estado de los elementos* de la misma, y determina si se ha producido un fallo o si se trata de un fallo en potencia.

Ante un fallo o un fallo en potencia, Mantenimiento on-line dictamina las posibles causas, las soluciones alternativas y la disponibilidad de un intervalo de actuación. Si cuenta con un intervalo de actuación, se lo comunica al Scheduler dinámico para obtener el momento más adecuado desde el punto de vista de producción. Si la actuación es inmediata, también realiza la comunicación con objeto de arrancar una replanificación. En ambos casos, comunica las máquinas que van a estar inhabilitadas y la duración de la indisposición.

Para realizar el diagnóstico dispone de información sobre el funcionamiento de los elementos que integran el sistema, tanto a nivel individual como en su conjunto. La recopilación de estos datos requiere un *análisis exhaustivo del sistema aplicando FMEA* (Failure Mode and Effects Analysis) *y FTA* (Fault Tree Analysis):

- Con el FMEA se especifican las funciones de cada elemento, los modos de fallo que implican el incumplimiento de cada función, los efectos del modo de fallo, las causas del modo de fallo, los medios de detección para cada causa y modo de fallo, y la frecuencia, gravedad y posibilidad de detección de cada causa y modo de fallo.

- El FTA es un modelo lógico y booleano basado en el FMEA, que representa las combinaciones paralelas y secuenciales que son la causa de la ocurrencia de un acontecimiento superior, estado no deseado y generalmente más tardío.

Arquitectura

La arquitectura genérica del módulo Mantenimiento on-line se presenta en la figura 3.6 y la funcionalidad de sus componentes se describe brevemente a continuación:

- **Monitorización continua.** Recoge los valores de los sensores instalados en determinadas máquinas, analiza dichos valores mediante técnicas adecuadas para el tratamiento de señales, y a partir de estos datos, *detecta las anomalías en el estado de los elementos.*

- **Diagnóstico.** *Determina si se ha producido un fallo o existe un fallo potencial* en el sistema, indica las *causas de este suceso*, establece las *operaciones de mantenimiento* necesarias para solucionarlo, calcula la *duración* de estas acciones y especifica las máquinas que no van a estar disponibles.

- **Mantenimiento.** Gestiona la *ejecución del plan de mantenimiento* de acuerdo con producción (Scheduler dinámico), incluyendo en dicho plan las operaciones correctivas y/o predictivas generadas en el mantenimiento on-line.

- **Registro.** Guarda en un fichero *información sobre los fallos o tendencias de fallo* detectados, *y las operaciones de mantenimiento* realizadas por el operario.

- **Gestor de servicios y eventos.** *Procesa las peticiones de servicio y los eventos* que van a ser enviados, así como los mensajes que se reciban de otros módulos.

- **Interface con otros módulos.** Responsable del *intercambio de mensajes* con los restantes módulos del sistema de control.

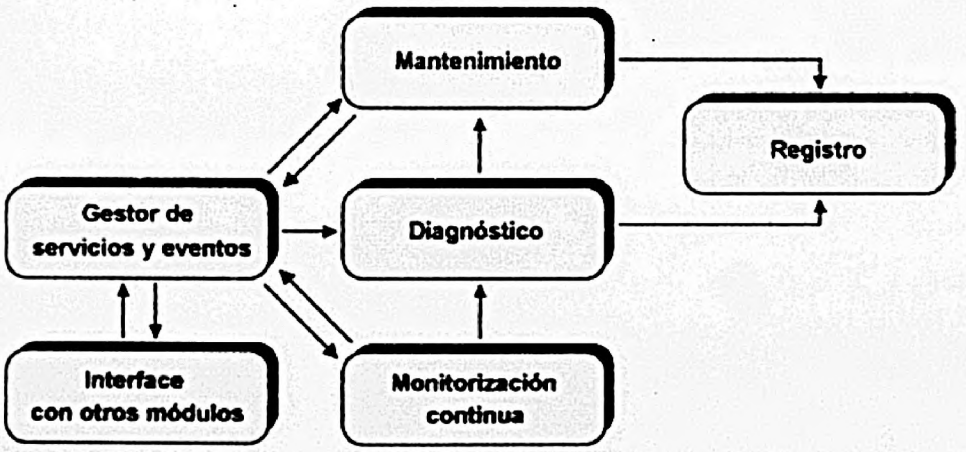


Figura 3.6: Arquitectura genérica del módulo Mantenimiento on-line.

3.2.2.2.3 Módulo Scheduler dinámico (on-line).

Funcionalidad

El Scheduler estático genera la planificación inicial de tareas óptima en modo off-line, considerando la situación de disponibilidad de los recursos principales.

El Scheduler dinámico recoge dicho plan y solicita una *imagen actualizada de la disponibilidad de recursos secundarios* (por ejemplo, herramientas en los almacenes y carruseles), con objeto de poder *replanificar las operaciones*, si fuese necesario. Posteriormente, el plan de producción se envía al CBD.

Durante la ejecución del plan, el Scheduler dinámico puede recibir eventos que informen de la aparición de imprevistos. Estos eventos son atendidos en orden de prioridad y fecha de aparición.

La búsqueda de una solución a los imprevistos consiste en un proceso iterativo con limitación temporal configurable. Aunque la disminución del intervalo de búsqueda vaya en detrimento de la calidad de la solución proporcionada, la necesidad de una respuesta rápida requiere llegar a un compromiso a la hora de asignar un valor.

Al procesar el evento, el módulo comprueba si el tiempo que resta para la finalización del plan es superior al plazo de búsqueda establecido. En tal caso, compensa efectuar la modificación. En caso contrario, desestima toda modificación y deja concluir el plan previsto con la repercusión que implique la circunstancia no atendida.

Si compensa la modificación, el CBD recibe una solicitud de no ejecutar ninguna nueva operación, y el Scheduler dinámico realiza las modificaciones on-line, siempre y cuando se justifique este coste frente a la regeneración. Sin esta justificación, el Scheduler dinámico solicita al estático una replanificación con las condiciones actuales.

El Scheduler dinámico transforma el plan de producción mediante una sencilla estrategia de recuperación según el evento recibido. Como ejemplos más representativos se indican los siguientes:

- *Cancelación de un pedido.* Libera los recursos asignados a dicho pedido, brindando la posibilidad de aprovecharlos para otras tareas.
- *Alta de un nuevo pedido.* Busca la mejor alternativa para la producción del pedido en el menor tiempo posible, estudiando las posibles rutas de fabricación.
- *Cambio de la prioridad de un pedido.* Intenta adelantar las operaciones del pedido en el tiempo. Si no puede, mantiene la planificación prevista.
- *Fallo o mantenimiento de una máquina.* Trata de desviar las operaciones planificadas para dicha máquina a otras alternativas. Si no fuera posible, deja las operaciones en estado pendiente hasta que se solvente el fallo.
- *Arreglo de una máquina.* Replanifica las operaciones en estado pendiente afectadas por el paro de dicha máquina.
- *Falta de material.* Todas las operaciones de piezas planificadas que requieran dicho material quedan en estado pendiente.
- *Aprovisionamiento de material.* Replanifica las operaciones en estado pendiente por la falta de dicho material.

Modificado el plan, el Scheduler dinámico se lo notifica al CBD para que proceda a su ejecución.

Arquitectura

La arquitectura genérica del módulo Scheduler dinámico se presenta en la figura 3.7 y la funcionalidad de sus componentes se describe brevemente a continuación:

- **Configurador.** Establece el *entorno operativo de información* y permite *configurar* los siguientes *parámetros:* calendario de trabajo, horas laborables al día, horizonte de planificación, objetivos de fabricación (plazos, ocupación de máquinas), tiempo máximo de respuesta a un evento, tiempo mínimo para una replanificación, límite de una reacción adaptativa, coste económico del retraso por unidad de tiempo, máquinas, recursos, piezas, procesos, etc.

- **Estimador.** Analiza el impacto de las modificaciones sobre el plan de producción y *decide si se genera un nuevo plan (Scheduler estático), se adapta el existente o no se realiza ninguna modificación.* En los dos primeros casos, emite una petición de servicio de parada al CBD.

- **Selector de estrategia.** Refina el análisis realizado previamente con información relativa a la disponibilidad de recursos secundarios críticos (herramientas, programas de control numérico) y *selecciona la estrategia más adecuada a la situación real de la planta en ese instante.*

- **Secuenciador reactivo.** *Modifica el plan de producción* en función del tipo de eventualidad, del conjunto de operaciones pendientes para la finalización del plan en curso, y del coste o grado de bondad del plan a obtener por esta vía.

- **Evaluador.** Genera un conjunto de *estadísticas sobre el funcionamiento del sistema* en virtud de las adaptaciones del plan de producción al contexto de la planta de fabricación.

- **Gestor de servicios y eventos.** *Procesa las peticiones de servicio y eventos* recibidos de otros módulos, así como los mensajes enviados al exterior.

- **Interface con otros módulos.** Responsable del *intercambio de mensajes* con los restantes módulos del sistema de control.

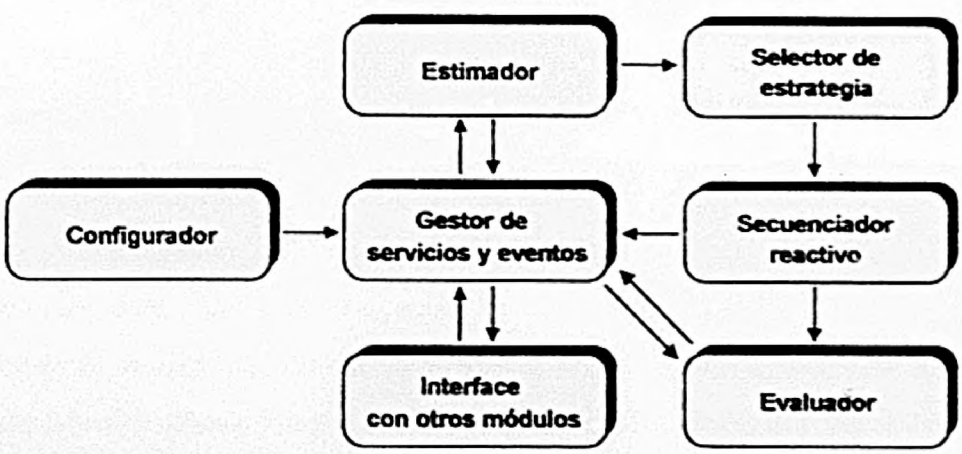


Figura 3.7: Arquitectura genérica del módulo Scheduler dinámico.

3.2.2.2.4 Módulo Control básico Dispatcher.

Funcionalidad

El CBD se encarga de *procesar el fichero con el plan de trabajo* (secuencia de piezas pertenecientes a diversas familias) suministrado por el Scheduler dinámico previa notificación. Este fichero contiene las tareas a realizar sobre las piezas, con información detallada de las operaciones de valor añadido, las operaciones de paletizado y las operaciones de transporte hacia el exterior de la planta de fabricación.

Cada operación tiene asociada una fecha de inicio planificada, la cual permite al CBD determinar el momento concreto de ejecución de la operación y mantener una secuencia ordenada de las tareas pendientes.

Sin embargo, para completar la ejecución del plan de trabajo es necesario realizar otro conjunto de tareas que no se recogen en el fichero: transportes intermedios, cargas en las máquinas, gestión de las herramientas, gestión de los programas de control numérico, descargas de las piezas, etc. Estas tareas se generan en *la ruta de fabricación genérica que establece el CBD.*

Asimismo es labor del CBD *la transformación de todas las tareas en peticiones de servicio a los restantes módulos* del sistema de control, sobre todo hacia los servidores de dispositivo y los gestores de recursos.

Normalmente, los servicios son independientes unos de otros, pero existen situaciones en las que la ejecución de una determinada tarea requiere la acción coordinada de distintos módulos. En tal caso, el CBD debe *sincronizar la ejecución de los servicios* en los módulos correspondientes.

Los eventos en respuesta a la ejecución de las peticiones de servicio y demás eventos informativos recibidos por el CBD deben ser procesados. Generalmente, este tratamiento supone la emisión de mensajes (peticiones de servicio o eventos) a otros módulos.

En cada sesión, todo mensaje recibido o enviado por este módulo queda registrado en el *fichero de incidencias*, con la siguiente información: la fecha, la hora, el remitente o destinatario, y el tipo de mensaje (servicio o evento, e identificador).

Arquitectura

La arquitectura genérica del módulo CBD se presenta en la figura 3.8 y la funcionalidad de sus componentes se describe brevemente a continuación:

- **Dispatcher.** Está constituido por:
  - **Configurador.** Crea una *imagen de la planta de fabricación* a partir de la información de estado de todos los dispositivos, y realiza las operaciones de inicialización de los distintos componentes.
  - **Procesador del plan.** Tras la lectura del fichero con un nuevo plan de trabajo, *solicita la carga de las herramientas, los programas de control numérico y los programas de robot necesarios para fabricar el plan*, aprovechando al máximo la capacidad de los dispositivos y minimizando el número de cargas posteriores. También realiza las acciones necesarias en caso de una replanificación.
  - **Generador de tareas.** Establece el *conjunto de operaciones intermedias* que unidas con las recogidas explícitamente en el plan, permiten la fabricación de las distintas piezas.
  - **Conversor.** *Transforma las tareas en peticiones de servicio y/o eventos* hacia los distintos módulos que componen el sistema de control.
  - **Registro.** Deja constancia de todo mensaje recibido o enviado por este módulo en el *fichero de incidencias*.
  - **Gestor de servicios y eventos.** *Procesa las peticiones de servicio y los eventos* que van a ser enviados, así como los mensajes que se reciban de otros módulos.
  - **Interface con otros módulos.** Responsable del *intercambio de mensajes* con los restantes módulos del sistema de control.

- **Asignador de recursos.** *Sincroniza la ejecución de aquellos servicios cooperantes* necesarios para la realización de una tarea que involucra distintos módulos.

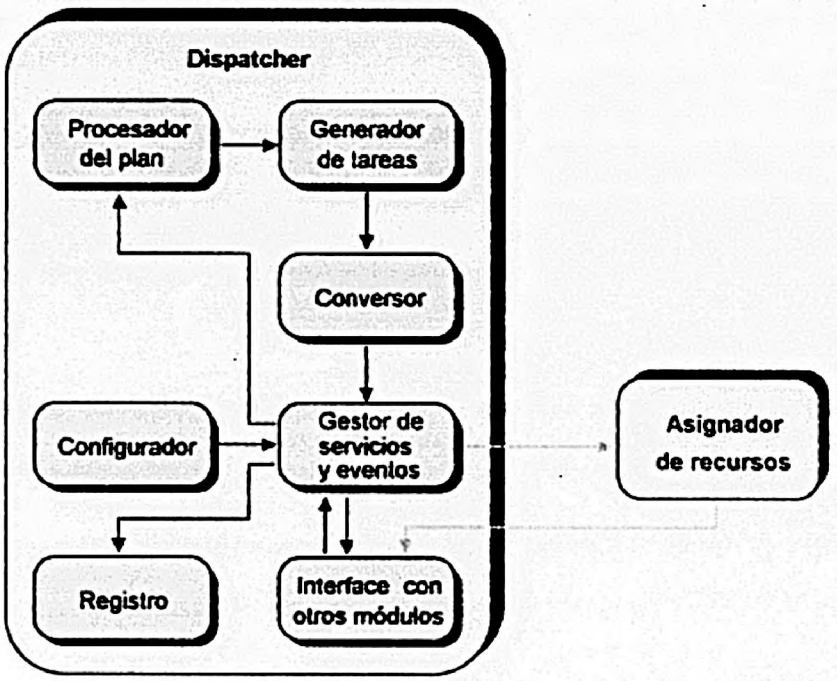


Figura 3.8: Arquitectura genérica del módulo Control básico Dispatcher.

3.2.2.2.5 Módulos gestores de recursos y servidores de dispositivos.

Funcionalidad

Un servidor de dispositivo es fundamentalmente un ejecutor de comandos sobre los controladores de planta. Convierte las solicitudes de servicio procedentes de otros módulos en un conjunto de acciones inteligibles por dichos controladores, los cuales proceden con su ejecución.

Finalizada la ejecución de la tarea, el servidor se encarga de comunicarlo a los correspondientes módulos del sistema de control. De esta forma, *proporciona una imagen de lo que está sucediendo realmente en el sistema físico.*

Cuando la complejidad de la gestión del recurso excede el ámbito de los servidores, son necesarios los gestores de recursos. Las situaciones más comunes que precisan de estos módulos son las siguientes:

- Existen varios dispositivos de planta que pueden realizar una misma función y compartir recursos (por ejemplo, selección de un AGV para un transporte).
- Existen recursos de difícil gestión asociados o no con dispositivos (por ejemplo, selección de herramientas, de programas de control numérico, etc.).
- Hay que sincronizar varios servidores para satisfacer determinada funcionalidad (por ejemplo, robots cooperantes).
- Hay que supervisar o coordinar un área que funciona en parte de forma autónoma (por ejemplo, un PC que gobierna una célula de soldadura).
- Hay que gestionar información que atañe a varios módulos del sistema, o flujos de información a los cuales es necesario seguirles la pista.

Como regla general, *cada módulo gestiona los datos referentes a la parte de la planta que está bajo su responsabilidad y que no está gestionada por ningún otro módulo.* Así, el servidor mantiene la información relativa al dispositivo de planta que gobierna y su entorno, mientras el gestor mantiene la información que es genérica a los servidores que controla y a los recursos que gestiona.

Por lo tanto, los servidores y gestores son módulos especializados en áreas específicas de la aplicación, relegando de dichas labores al CBD. De este modo, *se obtiene un control distribuido pero con una imagen homogénea de la gestión,* no sólo de cada dispositivo sino también de sistemas de adquisición de datos, e incluso del propio operario (se modela su comportamiento mediante un servidor).

Arquitectura

La arquitectura genérica de los módulos servidores se presenta en la figura 3.9 y la funcionalidad de sus componentes se describe brevemente a continuación:

- **Interface con otros módulos.** Responsable del *intercambio de mensajes*. Normalmente, se reciben peticiones de servicio de otros módulos y se envían las respuestas correspondientes. Ocasionalmente, se envían eventos para informar de algún suceso de interés, e incluso, peticiones de servicio hacia otros módulos.

- **Gestor de servicios y eventos.** *Procesa las órdenes recibidas* de otros módulos, *y genera las respuestas* a dichas peticiones de servicio y demás mensajes enviados hacia el exterior.

- **Driver.** Está constituido por el núcleo del driver y el espía:

  - El **núcleo** *realiza la implementación del protocolo* de comunicaciones, la traducción de las peticiones de servicio a instrucciones inteligibles por el controlador del dispositivo, el envío de las mimas, y la comprobación de su correcta transmisión por la línea.
  - El **espía** es un componente requerido con determinados protocolos que no permiten la transmisión de eventos por parte del dispositivo. En tal caso, el espía *está continuamente solicitando el estado del dispositivo* al núcleo del driver, con el propósito de conocer cualquier cambio de estado del mismo.

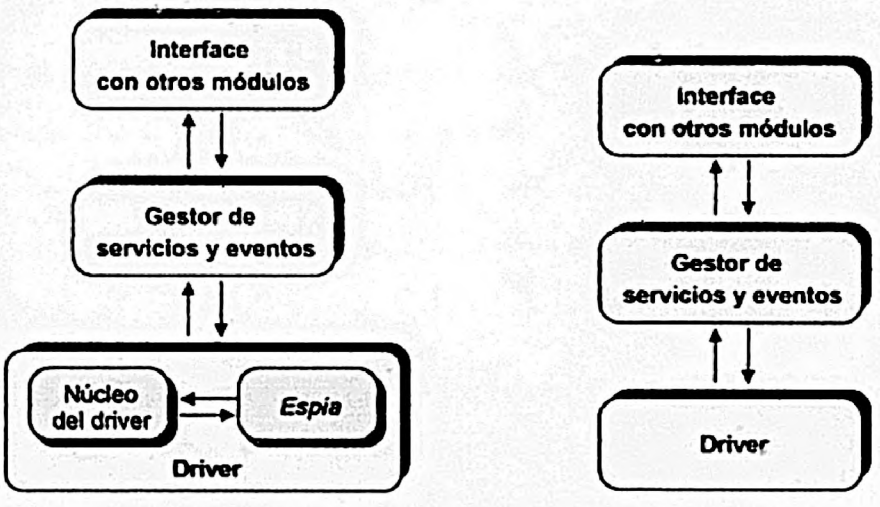


Figura 3.9: Arquitectura genérica de los módulos servidores y gestores respectivamente.

La arquitectura genérica de los módulos gestores de recursos se presenta en la figura 3.9. La única diferencia respecto a la arquitectura genérica de los servidores de

dispositivo reside en el componente driver: *en los gestores, el driver resuelve los accesos a una base de datos de recursos, mientras que en los servidores, este componente resuelve las particularidades de un dispositivo.*

3.2.2.2.6 Bus de mensajes.

Funcionalidad

El bus de mensajes es un software especializado en las tareas de mensajería entre procesos. Los módulos que constituyen el sistema de control envían los mensajes que desean transmitir a este bus, el cual se encarga de entregarlos en el destino especificado de forma transparente para el emisor. Esto significa que todos los módulos conectados al bus de mensajes pueden interoperar sin necesidad de conocer los detalles sobre la forma en que se efectúa realmente la comunicación.

En este sentido cabe destacar la labor del grupo OMG (Object Management Group) en la estandarización de las arquitecturas de sistemas de eventos orientados a objeto: CORBA (Common Object Request Broker Architecture) *[Mowbray, 95]*.

CORBA estandariza los servicios de gestión de objetos a través de plataformas heterogéneas, establece unas utilidades comunes, proporciona interfaces estándares de objetos que incluyen operaciones y parámetros, y presenta un modelo estándar para los gestores de eventos ORB (Object Request Broker). Todo ello revierte en un potente mecanismo de reutilización, y en concreto, el modelo de objetos fomenta una cuantiosa reutilización de objetos generales.

Existen en el mercado productos comerciales[23] *que implementan total o parcialmente el bus de mensajes, conocidos genéricamente como plataformas de integración y middlewares.* Estos productos constituyen no sólo un importante *mecanismo para integrar componentes reutilizables, sino que son en sí mismos componentes*

[23] Algunos ejemplos son: DAE (Distributed Automation Edition) y MQSeries de IBM, Basestar y DECMessageQ de DEC, SoftBench de HP y Orbix de IONA

reutilizables base para las comunicaciones en sistemas de eventos y arquitecturas de procesos comunicantes. Generalmente, son APIs (Application Program Interfaces) que proporcionan una interface programable sencillo entre plataformas, transmisión de mensajes en redes con protocolo TCP/IP, LU 6.2, IPX, etc., y comunicación tanto sincrona como asincrona.

Sin embargo, la mera utilización de un producto software que implemente *un bus de mensajes no garantiza la plena conexión* de todas las aplicaciones. Existen módulos con determinadas particularidades que precisan de componentes software (bridge o gateway) para obtener dicha conectividad. Ejemplos de ello se encuentran en módulos instalados en equipos que no están contemplados por el producto que implementa el bus, en equipos que no soportan el sistema de comunicaciones que requiere, o en módulos desarrollados con anterioridad imposibles de conectar directamente al bus.

Arquitectura

La arquitectura genérica del bus de mensajes se presenta en la figura 3.10 y la funcionalidad de sus componentes se describe brevemente a continuación:

- **Protocolo de mensajería.** Define la *interface de las peticiones de servicio y de los eventos:* identificadores, parámetros y demás declaraciones necesarias para establecer un consenso en el modo de comunicación de los módulos.
- **Bridge o Gateway.** *Resuelven las particularidades* de determinados módulos que no pueden conectarse al bus de mensajes de forma directa.
- **Librerías.** *Aislan los módulos conectados del bus de mensajes concreto* que se implemente. Esto significa que si se cambia el bus, no es necesario modificar los módulos, simplemente hay que cambiar las librerías.
- **Plataforma de integración o middleware.** Resuelve el problema de *comunicación entre procesos de forma transparente* para aquellos módulos que estén conectados.

- **Protocolo de red.** Establece las *reglas que permiten iniciar, mantener y finalizar una transmisión de datos* a través de la red subyacente. En conjunto, protocolo y red constituyen el sistema de comunicaciones.

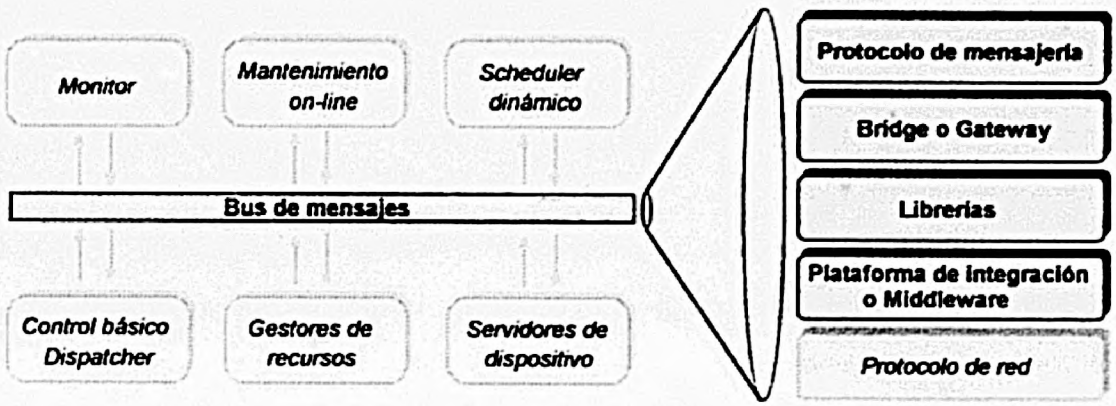


Figura 3.10: Arquitectura genérica del Bus de mensajes.

3.2.2.2.7 Sistemas de comunicaciones industriales.

Funcionalidad

Los sistemas de comunicaciones *proporcionan los medios para integrar los distintos módulos del sistema de control, y conectar éstos con los dispositivos de planta.*

El modelo de referencia OSI (Open System Interconnection) de ISO (International Standards Organization) define un conjunto de mecanismos para permitir la interconexión de sistemas informáticos heterogéneos (figura 3.11).

Productos OSI para fabricación

Como productos diseñados y desarrollados en el marco del modelo OSI para los entornos de fabricación destacan:

- *MAP* (Manufacturing Automation Protocol). Especifica los protocolos funcionales de red para el entorno de planta.
- *CNMA* (Communications Network for Manufacturing Applications). Especifica, implementa, valida y promueve los estándares del modelo OSI para fabricación.

- *MMS* (Manufacturing Messaging Specification). Este protocolo del nivel de Aplicación OSI soporta la comunicación desde y hacia los dispositivos de planta.

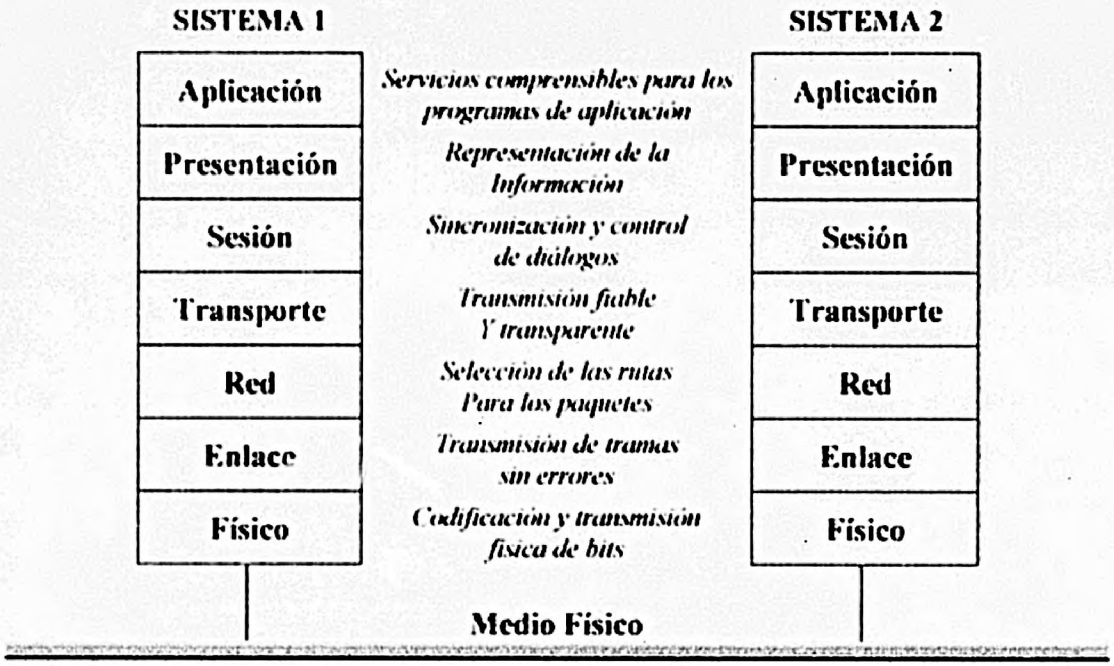


Figura 3.11: Modelo de referencia OSI para la interconexión de sistemas abiertos.

Estándares con mayor difusión

Otros estándares de interés, sin ser específicos para este entorno, son:

- *TCP/IP* (Transmission Control Protocol / Internet Protocol), que permiten la compartición de recursos a través de una red.
- *RPC* (Remote Procedure Call), que facilita la implementación de aplicaciones distribuidas.

Conexión con los dispositivos de planta

La conexión con los dispositivos de planta se realiza por *comunicación serie RS-232*, y en menor medida, mediante redes y buses de campo. En este último caso no existe un estándar como tal, y las principales tendencias que compiten son: FIP (Factory Instrumentation Protocol) y PROFIBUS (PROcess FIelBUS).

3.2.2.3 Verificación de las arquitecturas genéricas del dominio y de los módulos.

A continuación *se comprueba la trazabilidad de los componentes de las distintas arquitecturas en los requisitos funcionales particulares.* La consecución total de los requisitos no funcionales sólo se puede constatar finalizada la fase de implementación.

Verificación de la arquitectura genérica del dominio

La arquitectura genérica del dominio es consistente con la especificación del mismo, porque existe trazabilidad de los componentes de la arquitectura en el conjunto de requisitos funcionales (tabla 3.8).

Módulo	***Requisitos funcionales satisfechos***
Monitor	Monitorización, interface de usuario, gestión de la información y control del proceso estadístico
Mantenimiento on-line	Mantenimiento, gestión de errores y situaciones anómalas, gestión de la información y control del proceso estadístico
Scheduler dinámico	Planificación de las operaciones de planta, gestión de la información y control del proceso estadístico
Control básico Dispatcher	Coordinación general de las operaciones de planta, gestión de la información y control del proceso estadístico
Gestores de recursos	Gestión de la información y según casos, gestión de herramientas, de programas y/o de otros recursos
Servidores de dispositivos	Gestión de la información, gestión de piezas, de palets, de programas, de herramientas y de transporte
Bus de mensajes	Integración de los módulos

Tabla 3.8: Verificación de la arquitectura genérica del dominio.

Verificación de las arquitecturas genéricas de los módulos

Asimismo, *las arquitecturas genéricas de los distintos módulos son consistentes con sus respectivas especificaciones* (tablas 3.9-3.14).

Componente	*Requisitos funcionales satisfechos*
Configurador	Cambio del modo de funcionamiento y servicios de configuración
Gestor peticiones usuario	Interface con el usuario
Ayuda	Interface con el usuario (apoyo on-line)
Gestor servicios y eventos	Tratamiento de mensajes
Registro	Control del proceso estadístico
Actualizador	Monitorización
Representación gráfica	Monitorización
Interface otros módulos	Comunicación con otros módulos

Tabla 3.9: Verificación de la arquitectura genérica del módulo Monitor.

Componente	*Requisitos funcionales satisfechos*
Monitorización continua	Diagnóstico
Diagnóstico	Diagnóstico y terapia
Mantenimiento	Ejecución del plan de mantenimiento de acuerdo con producción
Registro	Control del proceso estadístico
Gestor servicios y eventos	Tratamiento de mensajes
Interface otros módulos	Comunicación con otros módulos

Tabla 3.10: Verificación de la arquitectura genérica del módulo Mantenimiento on-line.

Componente	*Requisitos funcionales satisfechos*
Configurador	Servicios de configuración
Estimador	Adaptación del plan de trabajo (decisión)
Selector de estrategia	Adaptación del plan de trabajo (selección de estrategia)
Secuenciador reactivo	Adaptación del plan de trabajo
Evaluador	Control del proceso estadístico
Gestor servicios y eventos	Tratamiento de mensajes
Interface otros módulos	Comunicación con otros módulos

Tabla 3.11: Verificación de la arquitectura genérica del módulo Scheduler dinámico.

Componente	*Requisitos funcionales satisfechos*
Configurador	Servicios de configuración
Procesador del plan	Ejecución del plan de trabajo (procesamiento del fichero)
Generador de tareas	Ejecución del plan de trabajo (generación de operaciones secundarias)
Conversor	Ejecución del plan de trabajo (traducción de las operaciones en mensajes)
Registro	Control del proceso estadístico
Gestor servicios y eventos	Tratamiento de mensajes
Interface otros módulos	Comunicación con otros módulos
Asignador de recursos	Sincronización de servicios

Tabla 3.12: Verificación de la arquitectura genérica del módulo Control básico Dispatcher.

Componente	*Requisitos funcionales satisfechos*
Interface otros módulos	Comunicación con otros módulos
Gestor servicios y eventos	Tratamiento de mensajes
Driver	Resolución de las particularidades especificas de dispositivos, bases de datos, etc.

Tabla 3.13: Verificación de la arquitectura genérica de los módulos gestores de recursos y servidores de dispositivos.

Componente	*Requisitos funcionales satisfechos*
Protocolo de mensajería	Comunicación entre módulos (definición de las interfaces de mensajes)
Bridge o gateway	Comunicación entre módulos (conexión al bus)
Librerías	Comunicación entre módulos (independencia del bus de mensajes)
Plataforma de integración o middleware	Comunicación entre módulos (transparencia en la comunicación para los módulos conectados)
Protocolo de red	*Software de base del sistema de comunicaciones*

Tabla 3.14: Verificación de la arquitectura genérica del Bus de mensajes.

Posibles configuraciones de la arquitectura genérica del dominio

Realizada la verificación de las arquitecturas genéricas, es preciso comprobar que en la *arquitectura genérica del dominio se captura la variabilidad necesaria para incluir a todos los productos potenciales de la familia* de software de control para FMSs.

Después de un detallado análisis se han determinado como posibles configuraciones las que se ilustran en la tabla 3.15. La implementación más sencilla contempla únicamente el módulo Monitor, los servidores de dispositivos, el sistema de comunicaciones y el Bus de mensajes; este último aunque no sea estrictamente necesario, si es recomendable

para la obtención de algunos requisitos no funcionales. La configuración más compleja está dotada de todos los módulos que componen la arquitectura genérica del dominio.

Módulos	*Configuraciones*												
Monitor	X	X	X	X	X	X	X	X	X	X	X	X	X
Mantenimiento on-line										X	X	X	X
Scheduler dinámico						X	X	X	X	X	X	X	X
Control básico Dispat.		X	X	X	X	X	X	X	X	X	X	X	X
Asignador de recursos			X		X		X		X		X		X
Gestores de recursos				X	X			X	X			X	X
Servidores de disp.	X	X	X	X	X	X	X	X	X	X	X	X	X
Bus de mensajes	X	X	X	X	X	X	X	X	X	X	X	X	X
Sist. comunicaciones	X	X	X	X	X	X	X	X	X	X	X	X	X

Tabla 3.15: Posibles configuraciones de la arquitectura genérica del dominio.

3.2.2.4 Soporte a la ingeniería de aplicación.

La adaptación de la arquitectura genérica del dominio a la especificación del nuevo producto, proporciona la arquitectura específica del producto y las especificaciones de mensajería de los distintos módulos involucrados.

A continuación *se explica cómo se establecen la arquitectura específica de un nuevo producto y las especificaciones de mensajería de los módulos.*

Proceso para establecer la arquitectura específica del nuevo producto

Los enlaces explícitos existentes entre la especificación y la arquitectura del dominio (tabla 3.8) permiten identificar inicialmente qué partes variables de la arquitectura genérica deben emplearse para instanciar la arquitectura del producto concreto, manteniendo su consistencia interna y su estructura original.

Posteriormente se refina la identificación de los módulos que compondrán el sistema de control teniendo en cuenta los siguientes criterios:

- A cada dispositivo o, en ocasiones, a un grupo de dispositivos que trabajan conjuntamente, se le asocia un módulo servidor de dispositivo.
- A aquel recurso compartido de difícil gestión que exceda el ámbito de los servidores de dispositivos, se le asocia un módulo gestor de recursos.

Las relaciones entre los módulos de la arquitectura del nuevo producto se recogen en los MFDs de las tareas identificadas en la operativa del producto. Estos *MFDs se obtienen a partir de los correspondientes MFDs generados en la ingeniería de dominio*, teniendo en cuenta los módulos y elementos del sistema actual. Si fuese preciso incorporar alguna petición de servicio o evento, tales mensajes deberán acatar el estándar de mensajería entre procesos impuesto para el dominio.

Proceso para establecer las especificaciones de mensajería de los módulos

Conocida la arquitectura del nuevo sistema de control, se establecen las especificaciones de mensajería de los módulos. Para ello, *se recuperan del repositorio las generadas en la ingeniería de dominio, y, si es necesario, se adaptan con la información procedente de los MFDs de las tareas identificadas en la especificación del nuevo sistema.*

Otros aspectos que se definen en relación con la posterior implementación del nuevo sistema son los siguientes:

- El equipo informático concreto en el que se instalará cada módulo.
- El modo de acceso lógico de cada módulo: dirección de red, nombre del buzón (mailbox, socket), etc.
- Las restricciones en la implementación de los mensajes: tiempo mínimo y máximo de respuesta, número máximo de peticiones de servicio que pueden ser procesadas al mismo tiempo, posibilidad de encolar peticiones de servicio, etc.

Conclusión

La arquitectura del nuevo sistema se obtiene particularizando la arquitectura del dominio, y lo mismo sucede con los MFDs. Las arquitecturas de los módulos son las establecidas en la ingeniería de dominio; las posibles variaciones residen en el conjunto de los mensajes que pueden procesar y no en sus componentes.

La verificación de las arquitecturas de estos módulos se ha realizado en la ingeniería de dominio, mientras que *la arquitectura correspondiente al nuevo sistema debe ser verificada contra la especificación concreta de la instancia de la familia de productos.*

3.2.3 Implementación del dominio.

Las tareas principales que se realizan en esta etapa son:

- **Implementación del software.** Consiste en construir los módulos identificados en la arquitectura, desarrollar herramientas automáticas y componentes de apoyo al proceso de generación del software.
- **Pruebas del software.** Éstas incluyen pruebas unitarias de los distintos componentes software, pruebas de integración que se planifican sobre la base de las relaciones existentes entre estos componentes, y pruebas de verificación que se realizan al nivel funcional y documental.
- **Soporte a la ingeniería de aplicación.**

3.2.3.1 Implementación del software.

En la construcción de los componentes software correspondientes a los módulos identificados en la arquitectura genérica del dominio, *son especialmente importantes la adquisición o construcción de los mismos, la tecnología utilizada, los atributos de los propios componentes y su almacenamiento para una utilización posterior.*

A continuación se indican los aspectos que se han considerado en cada caso, en relación con la reutilización sistemática.

Adquisición o construcción

Los módulos, sus componentes y los elementos de apoyo se pueden obtener por distintos medios: *adquisición de productos comerciales o de libre distribución, desarrollo de los mismos bajo la premisa de reusabilidad, ingeniería directa o reingeniería.*

En la implementación del dominio concreta realizada para validar la presente metodología y descrita en el siguiente capítulo, los medios empleados han sido:

- *Reingeniería para la implementación de los módulos.* Los trabajos previos son un buen punto de partida de cara a la construcción de los nuevos componentes, permitiendo reutilizar el conocimiento, los procesos y la experiencia.
- *Desarrollo para reutilizar en el caso de elementos de apoyo.* Los componentes de prueba, las plantillas, los generadores de aplicaciones, etc. proporcionan valiosos mecanismos a la hora de automatizar el proceso de generación de software e incrementan el grado de reutilización obtenido.

Tecnología

Evidentemente, emplear las últimas tecnologías en orientación a objetos, entornos de prototipado, sistemas abiertos y estándares de interfaces de usuario gráficos, permite conseguir un mayor grado de reutilización.

Sin embargo, es importante que en la elección de la herramienta de implementación se llegue a un *compromiso entre la potencia conceptual, la eficiencia y la compatibilidad con los restantes trabajos (previos y actuales).* La implementación se puede llevar a cabo con un lenguaje convencional, quizás sacrificando algunas capacidades. *El problema no reside en la funcionalidad o la potencia de cálculo, sino en la expresividad, la comodidad, la protección contra errores y la mantenibilidad.*

En este sentido, Rumbaugh *[Rumbaugh, 96]* indica que los lenguajes orientados a objetos permiten que la escritura, el mantenimiento, y la extensión del código sea más sencilla y segura, aun con sus variaciones en lo concerniente al apoyo que ofrecen para los conceptos orientados a objeto más avanzados.

Sin desmentir a Rumbaugh, ya que indudablemente los lenguajes orientados a objeto efectúan tareas que en el caso de los lenguajes convencionales deben realizarse manualmente, se debe recalcar que esta afirmación se cumplirá siempre y cuando se tengan presentes estos aspectos, básicos además para obtener la reutilización.

En cualquier caso, es *imprescindible prestar especial atención al estilo de programación.* Un buen estilo permite maximizar los beneficios de la programación, provenientes en su mayoría de unos costes de mantenimiento y de mejora sumamente reducidos, y de la reutilización del código en proyectos futuros.

Atributos de los componentes

Para que sean reutilizables, todos los elementos que se construyan, sea cual sea la tecnología empleada, deben tener los siguientes atributos:

- **Ortogonalidad.** *Independencia del componente software con relación a su contexto.* El incumplimiento de este atributo implica la reutilización conjunta del elemento con su contexto, lo cual disminuye el grado de reusabilidad al requerir la validación del nuevo contexto cuando no coincida con el originario.
- **Adaptabilidad.** Facilidad para introducir un componente software en un contexto nuevo, requiriendo normalmente modificaciones en el código fuente. Este atributo depende de dos factores: la *legibilidad* y la *estructura.*
- **Portabilidad.** *Capacidad para reutilizar un componente software en distintas plataformas hardware.* Este es un aspecto de gran interés, sobre todo teniendo en cuenta que es más barato adquirir hardware que rehacer las aplicaciones.
- **Bajo coste de reutilización.** No tiene sentido reutilizar un componente cuando es más caro el proceso de buscarlo, entenderlo y adaptarlo, que desarrollarlo partiendo de cero; en tal caso no existe aumento de la productividad. *Los*

mayores beneficios se obtienen al reutilizar los esfuerzos de diseño y desarrollo, no únicamente el código.

- **Alta fiabilidad.** Otra característica importante de cara a obtener un software adecuado para la reutilización es la *existencia de pruebas y casos de pruebas, necesarios para confirmar que los componentes funcionan correctamente.*
- **Documentación explícita de apoyo al proceso de reutilización.** El conocimiento y comprensión del componente son fundamentales para llevar a cabo una correcta reutilización. Por ello, es preciso generar una documentación adecuada concurrentemente con el desarrollo del software. Dicha documentación debe *recoger explícitamente las suposiciones necesarias para el perfecto comportamiento del componente*, entre otras informaciones.

Almacenamiento

Todos los componentes resultantes de esta etapa se almacenan en un *repositorio* junto a los elementos obtenidos en etapas previas. Este repositorio, diseñado e implementado en la ingeniería de dominio (apartado 3.2.5), dispone de mecanismos de ayuda a la recuperación de los elementos almacenados de cara a su posterior utilización.

También *es factible que coexistan varios componentes con la misma funcionalidad*, de modo que sea mayor la variación a la hora de seleccionar la configuración más adecuada a las necesidades de una aplicación concreta.

3.2.3.2 Pruebas del software.

Para realizar las pruebas, primero se deben *definir los componentes y las relaciones entre componentes que se van a probar*, detallando las características concretas de prueba, los criterios de aprobación y suspensión, los riesgos y la planificación. Es decir, *todas las pruebas requieren el diseño de los correspondientes casos de prueba y procedimientos.* Estos últimos describen la secuencia de acciones, incluyendo los pasos de preparación, los pasos de ejecución propiamente dicha, y los pasos de análisis de resultados.

Pruebas unitarias

Las pruebas unitarias *se realizan siguiendo la técnica de caja blanca;* es decir, se diseñan sobre la base de las estructuras de control y los procedimientos. Este tipo de técnica permite hacer una comprobación exhaustiva, probando todos los caminos, las decisiones, los bucles hasta sus valores límite y las estructuras internas de datos.

Pruebas de integración

En las pruebas de integración se emplea una *estrategia de integración incremental ascendente.* Se comienza probando las dependencias y tramos inferiores de los caminos de control, y se van incorporando componentes de niveles superiores hasta completar el módulo. En este caso tiene gran importancia el cumplimiento de la planificación temporal establecida, puesto que la realización de ciertas pruebas requiere la disponibilidad de determinados componentes.

Pruebas de verificación

Las pruebas de verificación *se realizan al nivel funcional y documental.* Las primeras comprueban que el funcionamiento del software coincida con las especificaciones del módulo correspondiente, mientras que las segundas evalúan el proceso de desarrollo seguido a través de los documentos generados en las distintas fases.

Los casos de prueba que se aplican para la verificación de los requisitos funcionales son de tipo caja negra; es decir, se basan en el conocimiento de las funciones específicas que debe realizar el módulo correspondiente, tratando de comprobar que sean operativas en su totalidad. Además, dependiendo de la funcionalidad a probar, *se elige el método de prueba de caja negra más adecuado:*

- *Análisis del valor límite.* Comprobación de la funcionalidad en los extremos de los campos de entrada.
- *Partición equivalente.* Reducción del número de casos de prueba a conjuntos de pruebas equivalentes.

- *Grafos causa-efecto.* Representaciones de las distintas condiciones lógicas y acciones asociadas.

La ejecución de estas pruebas se efectúa en modo simulación; se prescinde de los restantes módulos de control, si bien están presentes todos los componentes del módulo en cuestión. De este modo se puede realizar una verificación exhaustiva y rápida, eludiendo los problemas del entorno operativo.

Componentes auxiliares para las pruebas del software

Normalmente, para la ejecución de estas pruebas se necesitan componentes auxiliares:

- **Drivers.** Simulan llamadas al objeto bajo prueba y lo excita con conjuntos de datos de prueba.
- **Stubs.** Simulan el comportamiento de componentes con los que está relacionado.

Estos componentes auxiliares y demás componentes de apoyo desarrollados también están sujetos a las pruebas especificadas en este apartado.

3.2.3.3 Soporte a la ingeniería de aplicación.

Con los módulos y componentes desarrollados en la ingeniería de dominio, se establece la *estrategia que gobierna el proceso de construcción de la instancia de la familia de producto.* Este proceso *se subdivide en otros dos: el proceso para construir los módulos del nuevo sistema y el proceso para construir el sistema final.*

Proceso para construir los módulos del nuevo sistema

Las arquitecturas genéricas van a guiar el proceso de construcción de los respectivos módulos. *Para cada uno de ellos, se identifican aquellos componentes que pueden ser reutilizados directamente, cuales deben ser adaptados y cuales son de nueva creación.*

Posteriormente, *se ensamblan todos los componentes* (reutilizados, adaptados y de nuevo desarrollo), y *se realizan las pruebas unitarias, de integración y de verificación*

de cada módulo. Estas pruebas, especificadas en la ingeniería de dominio, pueden requerir cambios cuando algún componente sea adaptado o de nueva construcción.

Proceso para construir el sistema final

El producto final se obtiene mediante la integración de todos los módulos identificados en la arquitectura concreta del sistema, y la realización de las pruebas necesarias.

Las *pruebas unitarias* de los distintos módulos que componen el sistema de control *coinciden con las pruebas de verificación funcional realizadas para cada módulo.*

Las pruebas de integración se realizan siguiendo la estrategia de integración incremental ascendente de las distintas áreas funcionales o workstations[24]. Esta división en workstations, también permite modelar el comportamiento de las mismas, obteniendo así un estudio pormenorizado del funcionamiento de la planta.

Las clases de workstations más habituales en un FMS de mecanizado por arranque de viruta son: workstation de paletizado, workstation de transporte, workstation de torno, workstation de centro de mecanizado y workstation de despaletizado.

En cuanto a *las pruebas de verificación del sistema, se realizan al nivel funcional y documental. La verificación funcional se lleva a cabo siguiendo la técnica de caja negra en modo simulación:* se comprueba el funcionamiento del software de acuerdo con las especificaciones del mismo, pero sin estar integrado en el entorno operativo real.

La validación final del sistema, en la que interviene el cliente, se realiza en modo entorno operativo. Pero como la ejecución de todas las pruebas realizadas en la etapa anterior puede resultar demasiado costosa, se precisa una cuidadosa selección de los conjuntos de pruebas a efectuar en el sistema, ya integrado en el entorno operativo real.

---

[24] Cada workstation se encarga de una serie de tareas de fabricación, para lo cual tiene asignados determinados equipos de planta y sus controladores. Estos equipos pueden estar compartidos por varias workstations, sin afectar a su funcionalidad. La disponibilidad de tales recursos únicamente afecta en el apartado de restricciones impuestas a cada una de las workstations.

Conclusión

En la construcción del nuevo sistema los **módulos involucrados se obtienen reutilizando o adaptando los construidos en la ingeniería de dominio. El sistema final consiste en la integración de estos módulos,** sujeta a un conjunto de pruebas de integración y verificación adecuadas al nuevo sistema, aunque con posibilidad de reutilizar o adaptar alguna de las existentes en el repositorio.

3.2.4 Gestión del dominio.

La gestión del dominio comprende la *planificación, seguimiento y control del esfuerzo de la ingeniería de dominio.* También engloba las facetas de *gestión de procesos para el dominio, incluidas las disciplinas de gestión de la configuración y aseguramiento de la calidad.* Asimismo, *proporciona soporte a las necesidades y prioridades de los proyectos de la ingeniería de aplicación con el fin de satisfacer al cliente.*

A continuación, todas *estas tareas se agrupan en las relacionadas con la coordinación de las actividades de la ingeniería de dominio y las que conforman el soporte a la ingeniería de aplicación.*

3.2.4.1 Coordinación de las actividades de la ingeniería de dominio.

La óptima ejecución de las tareas de planificación, seguimiento y control, garantía de la calidad y gestión de la configuración, asegura una correcta coordinación.

Planificación

Una de las actividades cruciales del proceso de gestión del dominio es la *planificación del desarrollo y evolución de la familia de productos.* Esta actividad conlleva la obtención de estimaciones del esfuerzo humano requerido, de la duración cronológica del proyecto y del coste del mismo, junto con un exhaustivo análisis de riesgos.

El resultado de esta actividad es *un plan del dominio*, que determina cómo se van a emplear los recursos para construir una familia de productos de alta calidad y un proceso eficiente para la ingeniería de aplicación.

Este plan *presenta la asignación de recursos* para cada una de las etapas de desarrollo y cada una de las tareas identificadas en las mismas, sin olvidar las tareas de formación del equipo de desarrollo y de documentación.

También *establece la secuencia temporal* del análisis, diseño e implementación del dominio. En esta última etapa, se ha considerado la implementación de los siguientes módulos: Monitor, Mantenimiento on-line, Scheduler dinámico, Control básico Dispatcher, dos gestores de recursos (para herramientas y programas de control numérico), cuatro servidores de dispositivos (para máquinas de control numérico, robots, autómatas programables y usuario), y Bus de mensajes. El desarrollo de cada uno de estos módulos y de los componentes de apoyo a la ingeniería de aplicación se puede hacer en paralelo.

Los criterios considerados a la hora de *verificar el plan* han sido que los objetivos sean realistas y contribuyan a conseguir los objetivos a largo plazo, que el plan sea completo y creible, y que los riesgos más importantes estén identificados.

Seguimiento y control

Una vez establecido el plan de desarrollo, comienza la actividad de seguimiento y control. *El seguimiento se lleva a cabo evaluando los resultados de todas las revisiones* realizadas en todo el proceso de ingeniería del software, y determinando si los hitos formales se han alcanzado en la fecha programada. *Las desviaciones importantes requieren la aplicación de los controles adecuados a fin de reducirlas con la mayor brevedad.*

Garantía de la calidad

También la garantía de la calidad exige ciertas normas a seguir a lo largo del desarrollo. Hay que tener presente una serie de requisitos no funcionales identificados

como objetivos de calidad, y garantizar el cumplimiento de los mismos mediante el uso de ciertas técnicas.

Los *criterios empleados para evaluar la calidad* del proceso de desarrollo del dominio en cada una de sus fases son: *la utilización de estándares, los modelos y técnicas de ingeniería del software, revisiones e inspecciones (revisiones de estado, revisiones de fase e inspección de elementos), y la documentación.*

Gestión de la configuración

Las actividades de gestión de la configuración están orientadas a *identificar, organizar y controlar los elementos generados* a lo largo del proceso de desarrollo del dominio, que están almacenados en el repositorio. Las *actividades principales* son las siguientes:

- *Garantizar el acceso* ordenado a la información de todos los componentes construidos para el dominio.
- *Seleccionar los objetos de configuración:* especificaciones, arquitecturas, diseños, codificaciones, pruebas, etc.
- *Definir la jerarquía de configuración.*
- *Establecer la identificación de los objetos de configuración.*
- *Estructurar el archivo del dominio,* que recoge, a modo de índice, toda la información correspondiente al dominio.

La gestión de la configuración se realiza mediante la *generación de configuraciones de referencia en las diferentes etapas del desarrollo.* Con la finalización de cada etapa, su configuración de referencia queda cerrada o bloqueada, es decir, protegida de cambios que no se realicen mediante los procedimientos formales establecidos: solicitud por escrito y aprobación del autor y revisor. *Todo cambio realizado queda reflejado* con los siguientes datos: la modificación, la fecha, el motivo y la descripción.

Los cambios que afectan a elementos de la etapa en curso, lo que se denomina configuración abierta, no requieren procedimientos formales. De este modo se pretende

flexibilizar el proceso de construcción del dominio, ya que hay que indicar a este respecto que el desarrollo es un proceso iterativo, con cambios y revisiones.

Finalmente, la realización de copias de seguridad periódicas permite salvaguardar todo este conocimiento de posibles negligencias o incidentes inesperados.

3.2.4.2 Soporte a la ingeniería de aplicación.

La ingeniería de dominio debe *asistir al director de un proyecto de ingeniería de aplicación en la identificación de las necesidades del proyecto y en la realización del plan del proyecto.*

A continuación *se indican las consideraciones derivadas de la ingeniería de dominio y que deben tenerse en cuenta en la ingeniería de aplicación* para asegurar la óptima utilización de los elementos del dominio.

Consideraciones en el plan del proyecto

Las tareas de gestión son complicadas debido a que un proyecto de construcción de software de control está habitualmente enmarcado dentro de un proyecto más amplio y con distintas partes involucradas (el cliente, el integrador de sistemas, los fabricantes de máquinas, los suministradores de equipos informáticos, etc.), con los *problemas de coordinación* asociados. Estos aspectos deben ser considerados en la planificación de la instancia de la familia de productos.

El plan debe contemplar la *asignación de recursos* para cada una de las etapas de desarrollo: análisis, diseño, e implementación de la aplicación, *teniendo presente aquellos elementos que pueden ser reutilizados directamente, aquellos que deben ser adaptados y aquellos que deben ser desarrollados.*

El procedimiento para la recuperación de componentes candidatos reutilizables es responsabilidad de la ingeniería de dominio, al igual que la correcta inclusión de nuevos componentes en el repositorio y la gestión de éste.

La planificación de la etapa de implementación debe realizarse considerando las workstations que se identifiquen para el proyecto concreto, *y planteando el desarrollo de los módulos de control de nivel inferior en primer lugar.* La adopción de este criterio permite estructurar y adelantar la ejecución de las actividades de prueba.

Consideraciones en garantía de la calidad

Las normas de garantía de la calidad impuestas en la ingeniería de dominio también están presentes en la ingeniería de aplicación, y *deben ser incluidas en los procesos de adaptación o nueva creación de componentes.* Los criterios empleados para evaluar la calidad del proceso de desarrollo de la aplicación son los estipulados en la ingeniería de dominio.

Consideraciones en gestión de la configuración

En lo que respecta a las tareas de gestión de la configuración se sigue la misma política. El *control de cambios* en configuraciones cerradas se debe realizar siguiendo los procedimientos formales, los cuales no son necesarios en el caso de configuraciones abiertas. También se hacen copias de seguridad periódicas.

Conclusión

La asistencia de la ingeniería de dominio en las tareas de gestión de un proyecto concreto **es imprescindible para asegurar la óptima utilización de los elementos del dominio.** También es de gran importancia esta asistencia en la definición de requisitos de nuevos sistemas, con objeto de **descubrir nuevos aspectos que puedan ser incluidos en los correspondiente incrementos de la evolución del dominio.**

Sin embargo, este último punto puede convertirse en un conflicto entre la ingeniería de dominio y la ingeniería de aplicación, ya que para la primera supone incrementar su oferta de componentes reutilizables en futuros proyectos, mientras que para la segunda constituye un riesgo más en el proyecto en curso.

3.2.5 Construcción del repositorio.

El repositorio debe *almacenar y gestionar todos los productos generados en las distintas actividades del proceso de ingeniería de dominio, así como los que en un futuro se construyan en el proceso de ingeniería de aplicación.*

Para tener una gestión efectiva de los productos almacenados *no necesariamente se requiere un repositorio automatizado.* De hecho, *inicialmente esta gestión se realizaba mediante un árbol de directorios.* Sin embargo, **un sistema de información que permita el acceso a toda la información almacenada y la manipulación de la misma simplifica en gran medida el proceso de reutilización de estos elementos.**

A continuación *se presentan los requisitos del repositorio, su diseño, las recomendaciones de implementación, su explotación y mantenimiento.*

3.2.5.1 Requisitos del repositorio.

Los requisitos funcionales, de interface y de seguridad para esta *herramienta de soporte al proceso de reutilización* se describen a continuación:

- **Manipulación de los elementos.** *Crear, consultar, modificar y borrar* los elementos almacenados en el repositorio, teniendo en cuenta que:
  - Todo elemento almacenado tendrá una cabecera estandarizada y unos atributos que permitirán su *identificación sin ambigüedad.*
  - Toda la información que se recupere debe *presentarse en el formato adecuado* (Word, Excel, PowerPoint, etc.).
- **Manipulación de las relaciones entre elementos.** *Establecer o eliminar* relaciones entre elementos, las cuales pueden ser de *tres tipos:*
  - *Librería.* Presenta toda la información del repositorio según la *estructura organizativa* que se establezca.
  - *Referencia.* Identifica la *cita de un documento* dentro de otro. Para garantizar la integridad de la información, si se borra o renombra el

documento referenciado, hay que efectuar las modificaciones oportunas en los documentos que lo citen.

- *Traza.* Relaciona un conjunto de elementos que siguen un *orden lógico,* por ejemplo: un item[25] de código no puede preceder a un item de diseño de su mismo módulo y componente.

- **Búsqueda de elementos.** *Cada nivel de clasificación debe disponer de un índice* que contenga una entrada por cada elemento que exista en dicho nivel.

- **Gestión de la configuración.** *Control de las diferentes versiones de un elemento,* para garantizar la recuperación de información ante posibles equivocaciones o consultar la evolución de un elemento concreto.

- **Querys.** Consultas a la base de datos mediante la *introducción de una palabra clave,* reduciendo los tiempos de búsqueda cuando el usuario no esté familiarizado con la organización de la información contenida en el repositorio.

- **Interface de usuario.** La aplicación dispondrá de *una interface de usuario amigable.* Preferiblemente, la interacción del usuario con la base de datos requerirá simplemente el ratón.

- **Control de accesos.** Únicamente aquellos *usuarios que estén autorizados* pueden acceder o manipular el contenido de la base de datos.

3.2.5.2 Diseño. Ítems de clasificación.

El repositorio está organizado en cinco colecciones o niveles de clasificación:

- **Nivel superior de control.** Organiza toda la información correspondiente a la planta de fabricación, con los *módulos y componentes que afectan al funcionamiento global de la instalación.*

---

[25] Ítem es una denominación genérica para cualquier elemento que se almacene en el repositorio.

- **Nivel intermedio de control.** Clasifica toda la información referente a los módulos que actúan de intermediarios entre el CBD y los servidores de dispositivos, es decir, los *gestores de recursos.*

- **Nivel inferior de control.** Realiza la taxonomia de toda la información correspondiente a los módulos que proporcionan una imagen homogénea de los dispositivos de planta, es decir, los *servidores de dispositivos.*

- **Nivel de comunicaciones.** Clasifica toda la información de los *módulos involucrados en las tareas de comunicación.*

- **Nivel informativo.** Organiza cualquier *información de interés general,* como por ejemplo: articulos de revistas especializadas, normas, etc.

Además de estos cinco niveles de clasificación, existe un conjunto de items que se relacionan con los anteriores mediante relaciones de libreria (figura 3.12). La descripción de estos items se realiza a continuación:

- **Índice.** Presenta *una entrada por cada elemento* que existe en el nivel.

- **Histórico.** Contiene *versiones anteriores.* Cuando se modifica, borra o sustituye un elemento, se puede enviar la versión anterior al Histórico.

- **Planta.** Clasifica los módulos y demás *información referente al funcionamiento global de la instalación.*

- **Módulo.** Organiza un *módulo software.*

- **Componente.** Identifica un *componente de un módulo software.* Un componente se puede dividir a su vez en otro u otros componentes.

- **Diseño.** Incluye todos los *items de diseño:* especificación de requisitos, diseño de arquitectura y diseño detallado.

- **Código.** Contiene todos los *items de código:* programas fuente, includes, librerias, ejecutables, DLLs, procedimientos de compilación y linkage, etc.

- **Pruebas.** Engloba todos los *items de pruebas:* plan de verificación y validación, plan de integración, plan de pruebas unitarias, componentes de prueba, etc.

- **Manuales.** Recoge aquellos *documentos que explican el funcionamiento* de los módulos y componentes: manual de usuario, manual de instalación, etc.
- **Máquina.** Clasifica los módulos y demás *información correspondiente a un dispositivo de planta.*
- **Documento general.** Hace referencia a *documentos de tipo informativo.*

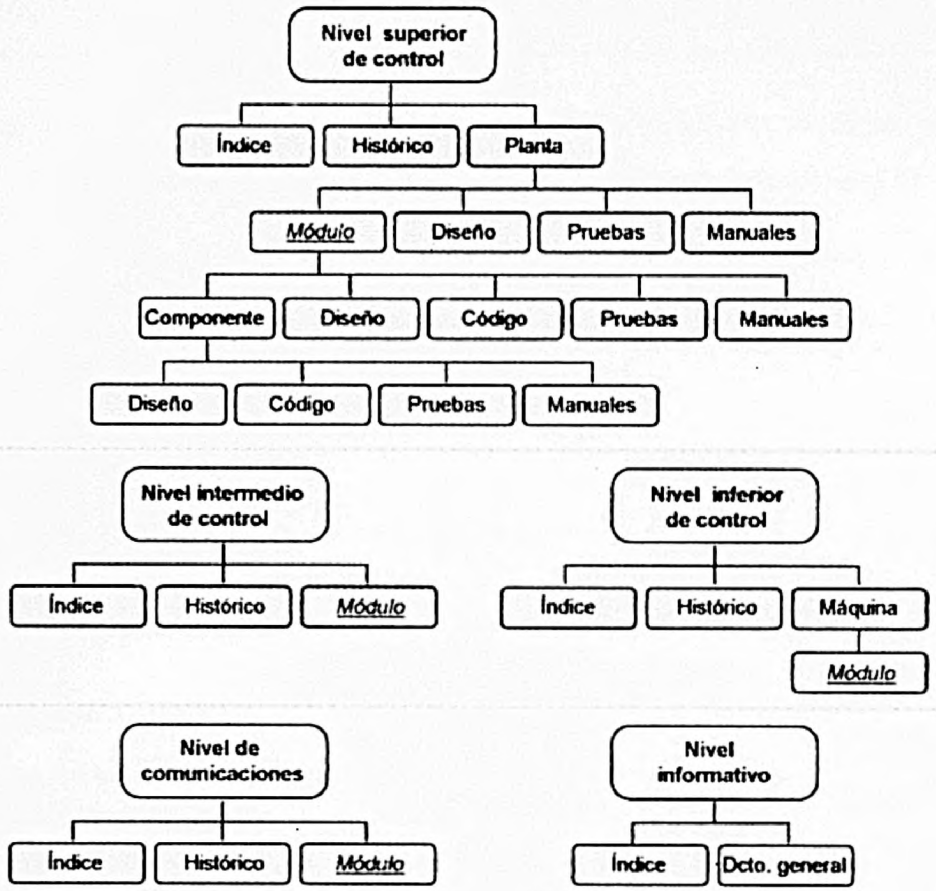


Figura 3.12: Relaciones de librería en los niveles de clasificación.

3.2.5.3 Implementación.

La aplicación de la tecnología orienta a objetos en las etapas de análisis y diseño, induce a elegir una *base de datos orientada a objetos como herramienta de implementación* del repositorio. Además, este tipo de base de datos *aporta importantes ventajas* en el desarrollo de estas aplicaciones frente las bases de datos relacionales.

Algunos de los inconvenientes más destacables de las bases de datos relacionales son los siguientes: las relaciones complejas deben expresarse en forma de código en el programa de acceso a la base de datos, únicamente se pueden agregar elementos a través de las tablas, los atributos de las tablas son públicos y de libre acceso, y los datos deben ser reducidos a primera forma normal.

3.2.5.4 Explotación y mantenimiento.

Todas *las etapas de construcción del repositorio son responsabilidad de ingeniería de dominio.* Su explotación se realiza en ambos procesos:

- *Ingeniería de aplicación accede al repositorio para buscar, seleccionar y reutilizar* los productos generados en el proceso de ingeniería de dominio o construidos en otros proyectos de ingeniería de aplicación.
- *Ingeniería de dominio se encarga de las tareas de mantenimiento y gestión* del repositorio (inclusión, modificación, borrado, etc.), y de proporcionar información actualizada sobre su organización y contenido para asistir en el proceso de ingeniería de aplicación.

Dada la importancia del repositorio, *esta herramienta debe estar sujeta a un proceso de mejora continua,* inspirado éste en la información procedente de las distintas instancias de la familia de productos y de la evolución del dominio.

3.3 Ingeniería de aplicación.

La ingeniería de aplicación es *similar al proceso de desarrollo de software convencional, pero está adaptado a los problemas y las necesidades de proyectos en un área de negocio concreta,* de forma tal que se obtiene una reutilización sistemática de los productos estandarizados en y entre proyectos de la familia de productos.

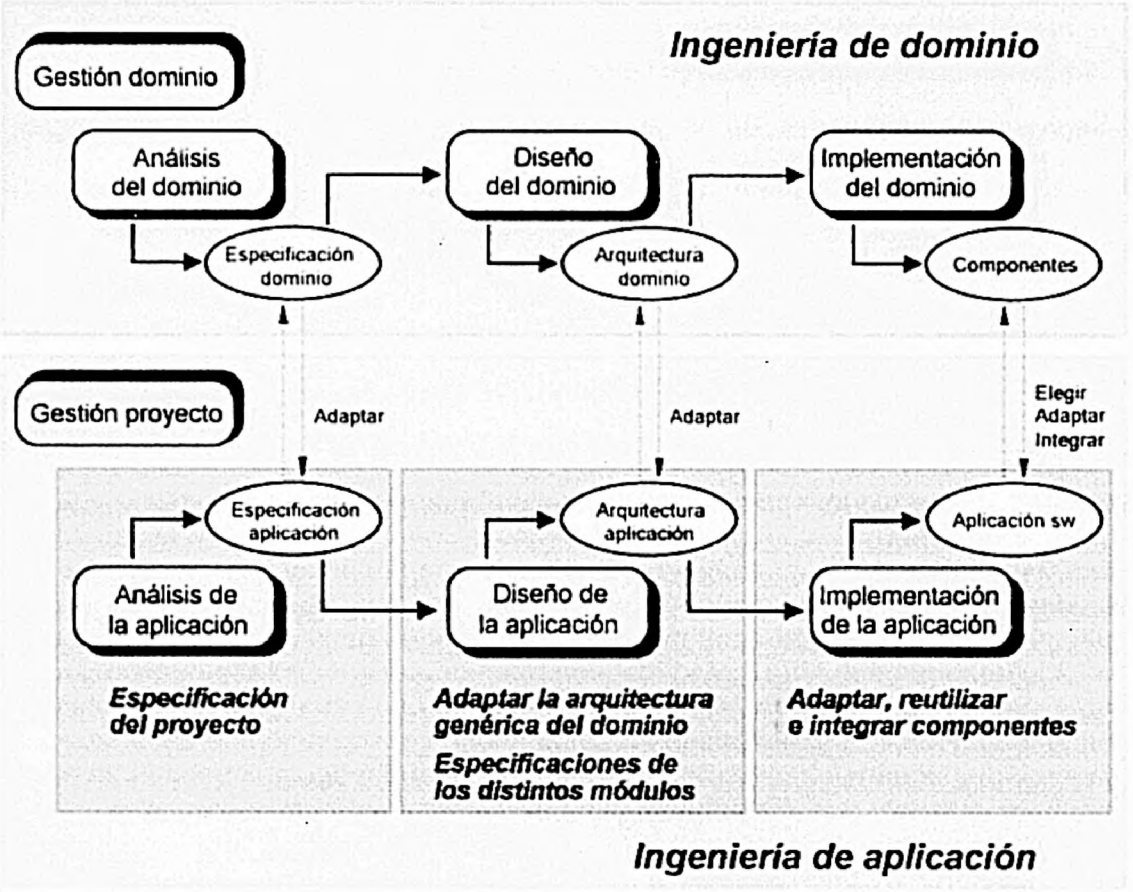


Figura 3.13: Actividades y productos de la ingeniería de aplicación.

Mientras la ingeniería de dominio determina el conjunto de sistemas que la ingeniería de aplicación puede construir, a esta última le concierne la construcción de un sistema de control concreto. Esto exige a la ingeniería de aplicación la realización de las actividades de *análisis, diseño, implementación y gestión del proyecto, según los procedimientos definidos en la ingeniería de dominio.* Para ello, dispone de la especificación del dominio, la arquitectura genérica de la familia de productos y de los

distintos módulos identificados, y un repositorio de componentes reutilizables específicos del dominio.

Para explicarlo detallamente, *se presentan a continuación el proceso y el desarrollo de un proyecto de ingeniería de aplicación.*

3.3.1 Proceso de ingeniería de aplicación.

Normalmente, los clientes son conscientes de la existencia de un problema vagamente comprendido y de la necesidad de una solución a dicho problema. En la ingeniería de aplicación se *establece una definición precisa del problema, con ayuda del conocimiento sobre el dominio, y se genera una solución.*

La especificación de la familia de productos permite identificar de forma precisa los requisitos del nuevo sistema. Posteriormente, se adapta la arquitectura genérica del dominio a la solución concreta, y se recuperan del repositorio, se adaptan o crean (si es preciso) y se integran los componentes necesarios, construyendo así el nuevo producto.

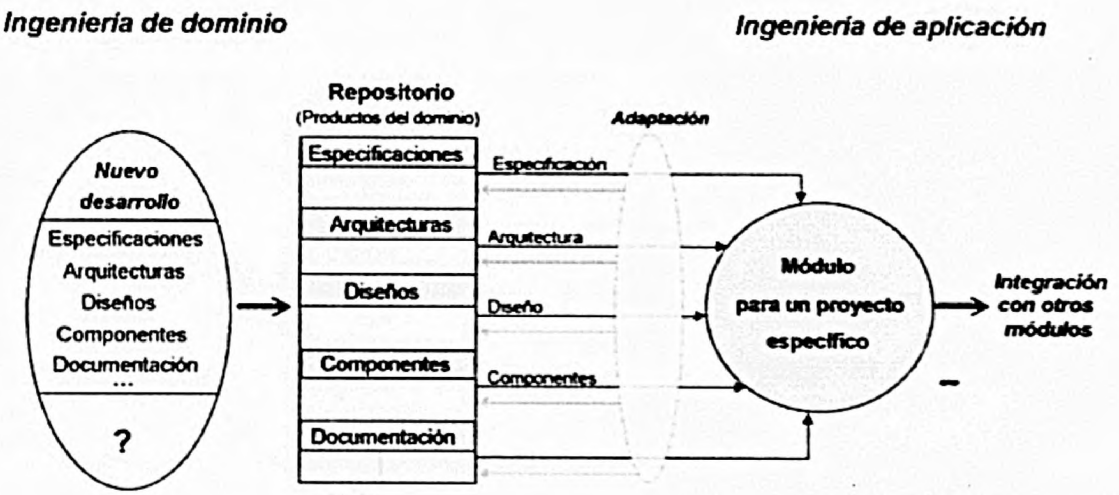


Figura 3.14: Proceso de ingeniería de aplicación.

Las prácticas de gestión difieren sustancialmente de las empleadas en los proyectos tradicionales. La causa de ello reside en el estado inicial de las actividades de diseño: de una hoja de papel en blanco se pasa a una selección y adaptación de componentes

dentro de una arquitectura genérica. *Las repercusiones inmediatas son: la reducción de la incertidumbre sobre la funcionalidad del sistema, y la reducción del esfuerzo requerido para construirlo.*

Conclusión

Resulta imprescindible que la ingeniería de dominio y la reutilización estén involucradas en todas las actividades de la ingeniería de aplicación. Las razones para ello son las siguientes:

- Si la especificación y el diseño se realizan partiendo de cero y se pretende reutilizar el código, muchos de los componentes (código) no se ajustarán a la especificación y el diseño. Al final esto se puede traducir en componentes inutilizables o en procesos de adaptación con mayor esfuerzo.
- La transición del análisis al diseño y la codificación conlleva un aumento considerable del volumen de información a tratar. El desconocimiento de tal volumen en un momento con presiones para completar el sistema, puede desalentar en la incorporación de la reutilización.
- Se precisa mayor integración entre la ingeniería de dominio y las actividades de desarrollo de sistemas, ya que normalmente suceden en paralelo.

3.3.2 Desarrollo de un proyecto de ingeniería de aplicación.

El proyecto de ingeniería de aplicación comienza cuando un cliente solicita el desarrollo de un sistema que encaja en la familia de productos ya construida. En este punto, la ingeniería de dominio ayuda a describir el problema en términos específicos del dominio y proporciona soporte en la identificación y uso de los componentes disponibles para la familia de productos.

La ingeniería de aplicación construye la *especificación* del nuevo sistema, la evalúa, la presenta al cliente y la revisa. Basándose en esta revisión y, en algunos casos, con el uso

de un prototipo exploratorio del producto, la ingeniería de aplicación descubre defectos adicionales u omisiones, lo cual conduce a la generación de una nueva especificación.

El siguiente paso es la construcción de la *arquitectura* específica del nuevo sistema. Su evaluación puede desvelar que alguna de las necesidades no se recoge en el modelo de forma apropiada. En tal caso, la ingeniería de aplicación lo consulta con la ingeniería de dominio para tomar una decisión sobre la inclusión de dicha necesidad en el modelo de la familia de productos.

La tarea de *evolucionar el dominio con la inclusión de nuevas necesidades o adaptación de las existentes, corresponde a la ingeniería de dominio.* Mientras tanto, la ingeniería de aplicación completa su modelo del sistema y genera un producto como solución aproximada a dicha necesidad, teniendo siempre presente la opinión del cliente.

Cuando la ingeniería de dominio finaliza su tarea de actualización, la ingeniería de aplicación revisa su modelo para obtener una *solución mejorada,* incorporando la utilidad añadida ya disponible.

Conclusión

Se puede concluir que **los nuevos proyectos de ingeniería de aplicación y la variación de las necesidades en los proyectos existentes, conducen a una revisión del consenso sobre el dominio.** Cada revisión proporciona nuevos componentes y nuevas guías de apoyo al proceso de desarrollo de los proyectos de ingeniería de aplicación. Esta evolución continúa hasta que la familia de productos no sea una línea de negocio viable.

3.4 Conclusiones.

La **nueva metodología para la construcción del software de control para FMSs, incorpora la reutilización de forma sistemática en todas las actividades del proceso de desarrollo,** incluida la de especificación.

El software así obtenido está dotado de los atributos de ortogonalidad, portabilidad, adaptabilidad, alta fiabilidad, bajo coste de reutilización y una documentación explícita de apoyo al proceso de reutilización. Además, **una reutilización sistemática permite beneficiarse totalmente de las ventajas de la reutilización:**

- *Productividad efectiva en el desarrollo del software.*
- *Obtención de sistemas técnicamente más fiables y fácilmente reconfigurables, con mayor funcionalidad, menores costes de desarrollo y de mantenimiento a lo largo de su vida útil, y menor tiempo de puesta en marcha.*

Pero el éxito de la metodología propuesta, se lograrán en la medida en que la ingeniería de aplicación interactúe fuertemente con la ingeniería de dominio. Existe una fuerte interacción entre ambas cuando la mayoría de los componentes empleados para desarrollar una aplicación particular residen en el repositorio, y es mínimo el esfuerzo requerido para una posible adaptación.

La metodología presentada tiene un doble interés: técnico y de aplicación práctica. Teóricamente, se introduce el uso de la ingeniería de dominio y la ingeniería de aplicación, se estudia la interacción entre ambas y se aporta una sistematización del procedimiento de desarrollo de un software reutilizable para el control de FMSs.

La aplicación práctica reportará beneficios a pequeñas y medianas empresas manufactureras, ya que *representa un gran paso en la resolución de dos graves problemas que acusa la construcción del software de control de FMSs:* el coste de desarrollo y el tiempo de puesta en marcha.

Capítulo 4

Proyecto piloto:

Construcción del software de control para un FMS

4.1 Introducción.

Para validar la metodología propuesta, se ha realizado un *proyecto piloto en el Departamento de Ingeniería Mecánica de la Escuela Técnica Superior de Ingenieros Industriales de Bilbao, en el bienio 1.996-97.*

El proyecto consiste en la implementación de los distintos módulos identificados en la arquitectura del dominio, bajo la premisa de reutilización, y la construcción de un proyecto de ingeniería de aplicación [Sarachaga, 97, 98]. Estos desarrollos van a permitir **evaluar la interacción entre la ingeniería de dominio y la ingeniería de aplicación, y por tanto, contrastar prácticamente el grado de bondad de la metodología propuesta.**

Dicho proyecto es la *culminación de una serie de fases previas,* en las cuales se ha variado el layout, la funcionalidad y los módulos de control *[Burgos, 93, 94], [Llorente, 93, 94, 94a, 94b, 94c]. Las adaptaciones, reestructuraciones y ampliaciones han permitido:*

- *Justificar la necesidad y la bondad de la metodología propuesta.* Como es sabido, la toma de decisión de automatizar una planta de producción es una decisión compleja por cuanto implica una fuerte inversión en equipamiento, reciclaje y cambio en la mentalidad del equipo humano. Tradicionalmente, la mayor parte de la inversión en equipamiento se lo llevaba el hardware, y un porcentaje bastante reducido el software.

  Actualmente está cambiando la tendencia sobre todo en los FMSs, donde el valor añadido reside especialmente en la flexibilidad para cambiar de una pieza a otra dentro de una misma familia y/o de una familia de piezas a otra.

  La mayor inversión para conseguirlo se concentra en el desarrollo de software reutilizable, y *una metodología con reutilización sistemática para el desarrollo del software de control de FMSs es la mayor aportación en este campo.* Aplicando la ingeniería de dominio y la de aplicación, esta metodología permite cambiar fácilmente de una familia de piezas a otra y de una pieza a otra dentro

de la misma familia. Esto redunda en un *aumento significativo de la flexibilidad del FMSs y la subsiguiente reducción de los costes y tiempos de desarrollo.*

- *Analizar el problema de selección de las herramientas para implementar el dominio.* El conocimiento y la experiencia adquirida a través de trabajos realizados anteriormente en este mismo proyecto piloto, han evidenciado que la reutilización es bastante pobre cuando las herramientas utilizadas son inadecuadas, pero sobre todo, cuando no existe una reutilización planificada, no se diseña para reutilizar y/o no se conocen las ventajas de la reutilización.

  Por esta razón se decidió implementar el dominio usando herramientas disponibles y conocidas, con la diferencia respecto a trabajos previos de involucrar la reutilización en todas y cada una de las etapas de desarrollo, y en el propio proceso de construcción.

Las adaptaciones, reestructuraciones y ampliaciones realizadas en relación con los trabajos anteriores y en al óptica de conseguir un alto grado de reutilización del software desarrollado, se presentan y analizan en el presente capitulo.

El capitulo se subdivide en **tres secciones:** *implementación del domino, ingeniería de aplicación y conclusiones.* La implementación del dominio y la ingeniería de aplicación son una aplicación de un proyecto específico de cuanto se ha analizado en el tercer capitulo.

4.2 Implementación del dominio.

La implementación del dominio consiste en:

- **Desarrollo de los distintos módulos y componentes identificados en la arquitectura genérica del dominio,** especificamente: *Monitor, Mantenimiento on-line, Scheduler dinámico, Control básico Dispatcher, dos gestores de recursos*[26] *(para herramientas y programas de control numérico), cuatro servidores de dispositivos (para máquinas de control numérico, robots, transporte y operario), y Bus de mensajes.*

- **Construcción de los componentes de pruebas y de otros componentes que automatizan o simplifican el proceso de ingeniería de aplicación.** Se han construido: los *drivers y stubs* necesarios para realizar las pruebas a estos módulos, y un *generador automático de bridges, librerías* y *plantillas* que simplifican el proceso de codificación de algunos componentes.

- **Construcción del repositorio y especificación de los procedimientos para la gestión y uso del mismo,** correspondientes a ingeniería de dominio e ingeniería de aplicación, respectivamente.

El esquema de implementación seguido para evaluar al mismo tiempo el grado de reutilización de los módulos construidos, aporta *un proceso de adaptación para un proyecto concreto de ingeniería de aplicación y los componentes de prueba para cada módulo.*

Asimismo, cuando ha sido necesario para incrementar la reutilización de un módulo, se ha propuesto una solución diferente de la adoptada en los trabajos previos al presente proyecto piloto.

---

[26] No se ha construido un gestor de programas de robot porque éstos no precisan modificaciones como los programas CN con las herramientas. Además, normalmente la memoria de un robot tiene capacidad para almacenar todos los programas necesarios, sino se almacenan en su servidor.

Las técnicas y herramientas empleadas en esta implementación se han seleccionado bajo el criterio de incrementar el grado de utilización. Así por ejemplo, para la construcción de los diagramas de flujo de mensajes (MFDs) se ha utilizado la herramienta Rational Rose *[Rational, 92]*, que soporta la metodología orientada a objetos de Grady Booch *[Booch, 94]*.

La tecnología aplicada en la construcción de los distintos módulos y componentes, y toda la información relevante para adaptarlos a productos concretos del dominio, se presenta en los apartados subsiguientes. El último apartado describe el almacenamiento de los componentes desarrollados en el repositorio y la construcción de este último.

4.2.1 Módulo Monitor.

4.2.1.1 Solución propuesta.

El sistema de monitorización anterior se habia generado con la aplicación *Plantworks [IBM, 92] bajo sistema operativo OS/2*, resultando la aplicación no portable y costosa de adaptar.

Para soslayar estos inconvenientes se han adoptado entornos más abiertos. El módulo Monitor construido reside en una *estación RS-6000 de IBM* con *sistema operativo AIX* y está implementado con el *lenguaje de programación C*. El entorno gráfico está realizado con *X-Windows*, software industrial estándar para la creación de interfaces gráficas. En un nivel superior se utiliza el conjunto de widgets (componentes de interfaces gráficas) *OSF/Motif*. Los componentes de comunicaciones utilizan la *libreria TCP/IP* del bus de mensajes.

También se ha desarrollado la *libreria para el manejo de ficheros*, formada por las funciones: *abrir un fichero, cerrar un fichero, escribir una cadena de texto, leer una linea, leer una palabra, leer un valor numérico, buscar una cadena en un fichero, leer*

hasta encontrar una cadena, leer un valor binario, y leer una cadena hasta un carácter separador.

Esta libreria ha sido sometida a un *exhaustivo plan de pruebas*, que utiliza las técnicas de caja blanca y las de caja negra. Los casos de prueba han sido minuciosamente elegidos, resultando una libreria fiable que se emplea en la implementación de otros módulos del sistema.

4.2.1.2 Proceso de adaptación.

Este proceso incluye los pasos a seguir para adaptar un módulo a una aplicación particular, una vez definida la especificación de mensajeria del módulo. En el caso del módulo Monitor estos pasos son:

- **Recuperar del repositorio el código del componente configurador.** Establecer el fichero que almacena la *configuración por defecto* y sustituir los identificadores del dominio por los correspondientes a la aplicación particular.
- **Recuperar del repositorio el código del componente gestor de peticiones de usuario.** Replicar aquellas peticiones de servicio a los dispositivos de planta reales necesarias para garantizar el funcionamiento semi-automático del sistema, y sustituir los identificadores de dominio empleados por los actuales.
- **Recuperar del repositorio el código del componente ayuda.** Adaptar la *tabla de relación* que asocia el tipo de ayuda con los ficheros que contienen dicha ayuda.
- **Recuperar del repositorio el código del componente gestor de servicios y eventos.** Eliminar todas aquellas funciones que implementen mensajes no identificados en su especificación, replicar aquellas funciones que solicitan la misma petición de servicio a varios módulos y/o reciben el mismo evento procedente de distintos módulos, y sustituir los identificadores del dominio por los particulares.

También hay que adecuar la *tabla de relación* que asocia cada mensaje con su código, el recurso emisor, su descripción, y las acciones que repercuten en las máquinas de control numérico, los robots, las posiciones del sistema de transporte y los distintos almacenes. Esta tabla se emplea en los componentes registro, actualizador y representación gráfica.

Este componente *incluye la interface con otros módulos*, que *precisa de la librería TCP/IP del bus de mensajes* para poder comunicarse con los restantes módulos.

- **Recuperar del repositorio el código del componente registro.** Identificar el *fichero* que almacena el código del dispositivo, el recurso emisor, la descripción, la fecha y la hora de los mensajes enviados y recibidos por este módulo.

- **Recuperar del repositorio el código del componente actualizador.** Particularizar los identificadores del dominio, y la *estructura con el estado global* de la planta que consta de los siguientes campos:

  - Modo de operación.
  - Fecha y hora de inicio de la sesión.
  - Datos de estado de cada máquina de control numérico: identificador del dispositivo y estado (activa, parada, error, mantenimiento).
  - Datos de estado de cada robot: identificador del dispositivo, estado (activo, parado, etc.), posición de giro y posición de desplazamiento.
  - Datos de estado de la ocupación de cada posición del sistema de transporte: identificador de la posición y estado (ocupada, libre, etc.).
  - Datos de estado de la ocupación de los almacenes de piezas, herramientas, garras, palets y utillajes: identificador del almacén, y estado del mismo (lleno, vacio).

- **Recuperar del repositorio el código del componente representación gráfica.** Construir la representación gráfica de la planta a partir de las rutinas gráficas para el trazado de los distintos elementos (centros de mecanizado, robots, etc.), y

adaptar el *fichero de cabecera* con las macros que caracterizan cada dibujo (dimensiones, posiciones, colores, etc.).

4.2.1.3 Componentes de prueba.

Para probar el módulo Monitor se emplea un *programa que simula el entorno*, evitando así poner en funcionamiento las otras aplicaciones del entorno de producción automático. Este software de simulación permite probar todos los componentes de módulo, incluida la interface de comunicaciones.

4.2.2 Módulo Mantenimiento (on-line).

Este módulo se desarrolla utilizando las herramientas FMEA (Failure Mode and Effects Analysis) y FTA (Fault Tree Analysis). La combinación de las mismas con la lógica Fuzzy *[Ayerbe, 96]*, asegura en la práctica la garantía de la calidad en el sector de la máquina herramienta y el modelado de la incertidumbre que puede encontrarse a la hora de trabajar con un FMS.

4.2.2.1 Solución propuesta.

Para el módulo Mantenimiento on-line se ha propuesto una arquitectura que utiliza el *protocolo TCP/IP* para comunicarse con los restantes módulos del sistema de control. Se ha *reutilizado el componente diagnóstico existente*. No se ha desarrollado uno nuevo porque tiene un bajo nivel de reutilización – debido a la dependencia de los dispositivos concretos de planta y la combinación de los mismos para efectuar las tareas de fabricación. La reutilización de este componente no impide probar el módulo en su conjunto y emplear dicho componente en el proyecto de ingeniería de aplicación.

La nueva arquitectura del módulo Mantenimiento on-line se ha codificado con las mismas herramientas que las empleadas en trabajos previos: *lenguaje de programación*

C y la herramienta *CubiCalc*[27] *[HyperLogic, 92] para implementar los árboles de fallo, y* se ejecuta en un *PC con sistema operativo MS-DOS y entorno Windows.*

Por último señalar que *no se ha modificado el módulo Mantenimiento off-line* porque está fuera del dominio en estudio.

4.2.2.2 Proceso de adaptación.

Conocida la especificación de mensajería del módulo, que se obtiene en la etapa de diseño del proceso de ingeniería de aplicación, su proceso de construcción consta de los siguientes pasos:

- **Desarrollar el programa de adquisición y tratamiento de señales para todos aquellos dispositivos con monitorización continua.** Estos *desarrollos específicos* deben generar todo mensaje de anomalía de acuerdo con la especificación de mensajería utilizada en el dominio. Dichos mensajes se transmiten por el bus de mensajes hasta el módulo Mantenimiento on-line.

- **Desarrollar el componente diagnóstico.** Obtener los FMEAs y FTAs (con sus grados de certeza) de la planta de fabricación, y construir la *base de conocimiento* en CubiCalc a partir de dicha información.

  También hay que adecuar el formato del *fichero de intercambio* con el componente mantenimiento, que debe contener tantos registros como fallos se detecten, junto con información sobre las causas y las certezas de las mismas.

- **Recuperar del repositorio el código del componente mantenimiento.** Generar el fichero que relaciona los distintos elementos con el fallo, la gravedad del fallo, las operaciones de mantenimiento y la duración. Todos estos datos se obtienen de los FMEA.

---

[27] CubiCalc genera un sistema experto basado en reglas pero desde una perspectiva Fuzzy.

Si es necesario, adaptar el fichero que relaciona la gravedad del fallo con un posible intervalo de actuación (período máximo de espera para realizar las operaciones de mantenimiento).

- **Recuperar del repositorio el código del componente registro.** Comprobar que el contenido del fichero es el deseado: identificador del caso, fecha, hora, operaciones de mantenimiento, elementos, resultado, duración, personal, materiales y herramientas.

  Este componente, diagnóstico y mantenimiento utilizan la librería de manejo de ficheros (ver el módulo Monitor).

- **Recuperar del repositorio el código del componente gestor de servicios y eventos, que incluye la interface con otros módulos.** Eliminar todos aquellas funciones que implementen mensajes no identificados en su especificación, replicar aquellas funciones que procesan el mismo evento procedente de distintos módulos, sustituir los identificadores del dominio por los de la aplicación particular, e incorporar las funciones necesarias para tratar todos aquellos mensajes relacionados con los sistemas de monitorización continua.

  Adaptar la *tabla de relación* que asocia los valores de las variables recibidos en los eventos, procedentes de los servidores y de los sistemas de monitorización continua, con las posiciones que ocupan en el fichero ASCII que necesita el componente diagnóstico para realizar la inferencia.

4.2.2.3 Componentes de prueba.

Las pruebas de los componentes mantenimiento y registro se realizan mediante una *interface de usuario* que permite solicitar operaciones de mantenimiento y realizar el reporte de las mismas.

Además, se dispone de un *programa que simula el envío de eventos* por parte de los servidores de dispositivos, el cual permite probar el comportamiento global del módulo.

Este software queda abierto a la inclusión de nuevas funciones que permitan realizar las pruebas de los mensajes enviados por los sistemas de monitorización continua.

4.2.3 Módulo Scheduler dinámico (on-line).

4.2.3.1 Solución propuesta.

A continuación se contrasta la solución adoptada para desarrollar el Scheduler dinámico con su predecesor *[Alvarez, 95]*.

Trabajos previos	***Scheduler estático***	***Scheduler dinámico***
Herramienta desarrollo	*Simfactory [CACI, 93]*	*G2[28] [Gensym, 92]*
Protocolo comunicación	*TCP/IP*	*GSI[29] y TCP/IP*
Equipamiento	*PC con MS-DOS y entorno Windows*	*Estación RS-6000 de IBM con AIX*
Solución propuesta	***Scheduler estático***	***Scheduler dinámico***
Herramienta desarrollo	*Witness[30] [Lanner, 96]*	*G2 [Gensym, 92]*
Protocolo comunicación	*OLE[31] y TCP/IP*	*GSI y TCP/IP*
Equipamiento	*PC con MS-DOS y entorno Windows*	*Estación RS-6000 de IBM con AIX*

Tabla 4.1: Solución anterior y solución propuesta para los módulos Scheduler.

---

[28] G2 es una herramienta orientada a objetos para generar sistemas expertos en tiempo real.

[29] GSI (G2 Standard Interface), que es la interface de comunicaciones de las aplicaciones G2.

[30] Witness es una aplicación Windows que permite representar la simulación del mundo real. Proporciona mecanismos para construir un modelo, ejecutar la simulación y recoger resultados, modificar el modelo, ejecutarlo de nuevo y comparar los resultados con el modelo original.

[31] OLE (Object Linking and Embedding) es un protocolo de Microsoft que permite la comunicación (linking) entre aplicaciones y el que una aplicación pueda manejar un objeto de otra (embedding).

4.2.3.2 Proceso de adaptación.

La construcción de un módulo Scheduler dinámico para una aplicación particular requiere la realización de las siguientes **tareas:**

- **Recuperar del repositorio la base de conocimiento de G2.** Modificarla para que contenga la *definición de las clases de objetos interface y objetos propios de la aplicación,* la declaración de las instancias de dichas clases, y las variables interface de datos entre aplicaciones con su asociación al objeto interface.

- **Recuperar del repositorio el código del componente configurador.** Adecuar los siguientes *ficheros* a las necesidades particulares:

  - *Fichero de configuración.* Establece el contexto de trabajo: días laborables, horas disponibles en el día, jornadas festivas, horizonte de planificación, tiempo máximo de respuesta a eventos urgentes, objetivos de fabricación, intervalo mínimo para una replanificación, coste del retraso de un pedido por unidad de tiempo, límite de una reacción adaptativa, e instante de comienzo de replanificación. Estos valores se cargan en las listas y variables de configuración.

  - *Fichero de dispositivos.* Permite establecer las dimensiones de las listas de huecos iniciales ordenadas por fecha, que acogen los huecos asociados a cada máquina con disponibilidad de la misma hasta el horizonte de planificación.

  - *Fichero del plan de producción*[32]. Contiene la siguiente información: identificador del pedido, identificador de unidad dentro del pedido, código Opitz, número de atada, identificador de la máquina, fecha de entrega, fecha mínima y máxima del inicio de la operación, fecha de inicio y fin real, tiempo de preparación y de producción, estado de la operación, y opcionalmente, información adicional y recursos auxiliares necesarios.

---

[32] El Scheduler dinámico envía este fichero sólo con las operaciones planificadas al gestor de herramientas para realizar la reserva de herramientas, y posteriormente, al CBD para que lo ejecute.

Cada operación del plan instancia un objeto de la clase operación, y cada recurso auxiliar instancia un objeto de la clase recurso.

- *Fichero de pedidos.* Almacena los siguientes datos: identificador del pedido, número de unidades, identificador del producto, fecha de entrega, estado del pedido, información adicional (opcional), y número de unidades pendientes de fabricar. Cada registro del fichero instancia un objeto de la clase pedido.
- *Fichero de rutas.* Contiene las posibles rutas de fabricación de cada producto. Cada registro instancia un objeto de la clase ruta asociado al producto.

- **Recuperar del repositorio el código del componente estimador.** Adaptar las siguientes *estructuras:*
  - *Tabla de relación* que asocia cada evento con un nivel de prioridad. Los eventos se atienden por orden de prioridad y fecha de creación.
  - *Fichero de pedidos para que el Scheduler off-line replanifique.* Tiene el mismo contenido que el fichero de pedidos comentado en el componente anterior.
  - *Fichero de la imagen instantánea de la planta.* El CBD envia la información relativa al estado de las máquinas, robots y buffers de almacenamiento intermedio.

- **Recuperar del repositorio el código del componente selector de estrategia.** Adecuar la *base de conocimiento* compuesta por el conjunto de reglas que enlazan las situaciones con las estrategias más adecuadas, y el fichero que notifica la fecha de inicio/fin de la replanificación y la estrategia de aplicación al Scheduler off-line.

- **Recuperar del repositorio el código del componente secuenciador reactivo.** Adaptar las *estrategias de tratamiento de los eventos:* alta de un pedido no planificado, cancelación/cambio de prioridad de un pedido planificado, fallo/arreglo de un dispositivo, y falta/reposición de material.

- **Recuperar del repositorio el código del componente evaluador.** Eliminar o adaptar *estadísticas* al conjunto existente sobre el funcionamiento del sistema en virtud de las adaptaciones del plan de producción al contexto de la planta. Las estadísticas implementadas son:
  - Resumen de pedidos planificados, pendientes y en curso, nivel de congestión y estado del sistema.
  - Porcentaje de utilización de la CPU, uso de la memoria, procesos en ejecución y mensajes, objetos e items.
  - Utilización de las máquinas y estado actual de cada una de ellas.
- **Recuperar del repositorio el código del componente gestor de servicios y eventos.** Eliminar todos aquellos procedimientos que implementen mensajes no identificados en su especificación, replicar aquellos procedimientos que procesan el mismo evento procedente de distintos módulos, y adaptar los siguientes *ficheros* al caso particular:
  - *Fichero de herramientas de carruseles.* Contiene los códigos de las herramientas disponibles en los carruseles, que se cargan en la lista de herramientas de carruseles.
  - *Fichero de herramientas de almacenes.* Contiene los códigos de las herramientas disponibles en los almacenes, que permiten actualizar la lista de herramientas de almacenes.
- **Recuperar del repositorio el código del componente interface con otros módulos.** Construir los *bridges* que permitan las interconexiones entre GSI/DAE[33] y GSI/OLE[34], tareas éstas que se realizan *con los generadores de bridges para G2 y para Witness.*

---

[33] El bridge GSI/DAE permite la comunicación entre el Scheduler dinámico y los restantes módulos de control conectados al bus de mensajes. DAE (Distribution Automation Edition) es la plataforma de integración que implementa dicho bus.

[34] El bridge GSI/OLE permite la comunicación entre el Scheduler estático y el dinámico.

4.2.3.3 Componentes de prueba.

Las pruebas se realizan mediante una *interface de usuario* que permite la generación manual de todos los eventos susceptibles de ser atendidos por este módulo. Las peticiones de servicio se retransmiten a un *software de simulación* que se encarga de visualizar la trama de información enviada.

4.2.4 Módulo Control básico Dispatcher.

4.2.4.1 Solución propuesta.

La solución adoptada es una *generalización del trabajo de López [López, 94, 95, 95a], [Llorente, 96, 96a].* El CBD construido en el presente proyecto piloto está implementado con la herramienta *G2 [Gensym, 92]* y emplea *GRAFCHART*[35] *[Arzen, 92]* para describir los componentes del sistema, su comportamiento, y los elementos de secuenciación y sincronización. Se ejecuta en una *estación RS-6000 de IBM* con *sistema operativo AIX*, y la comunicación con los restantes módulos se realiza a través de *GSI*.

Los *diagramas GRAFCET* son los que van a guiar el proceso de adaptación a las necesidades de un proyecto particular, una vez definida la especificación de mensajería del módulo. Estos diagramas y sus macros asociadas *se catalogan en cinco grupos:*

- *Diagrama principal.* Describe el funcionamiento general del sistema. Las acciones que se realizan en sus etapas son: el arranque de los diagramas de ruta para las piezas, la inclusión de objetos tarea al final de las colas de mensajes de los diagramas de las workstations, y la introducción de objetos servicio al final de las colas de mensajes de los diagramas correspondientes a los gestores y servidores.

---

[35] GRAFCHART es una implementación del lenguaje estándar GRAFCET IEC 848 sobre G2.

- *Diagrama de ruta de pieza.* Presenta una ruta genérica que se adapta a la secuencia de fabricación elegida por el Scheduler dinámico para cada pieza. Las acciones básicas que se realizan en sus etapas consisten en introducir objetos tarea al final de las colas de mensajes de los diagramas de las workstations.

- *Diagramas de workstations.* Describen el funcionamiento de las distintas workstations que pueden existir en un FMS: Workstation de paletizado, workstation de torno, workstation de centro de mecanizado, workstation de transporte, y workstation de despaletizado. Sus etapas transforman los objetos tarea en objetos servicio que incorporan al final de las colas de mensajes de los diagramas de los gestores, servidores y del asignador de recursos.

- *Diagramas de gestores de recursos.* Definen el funcionamiento de los gestores de recursos: gestor de herramientas y gestor de programas de control numérico. Sus etapas realizan el envío de peticiones de servicio a otros módulos.

- *Diagramas de servidores de dispositivos.* Representan el funcionamiento de los servidores y de los dispositivos bajo su control, y se distinguen cuatro tipos: servidor de control numérico, servidor de robot, servidor de transporte y servidor de operario. Como en los diagramas de gestores de recursos, sus etapas realizan el envío de las peticiones de servicio a otros módulos.

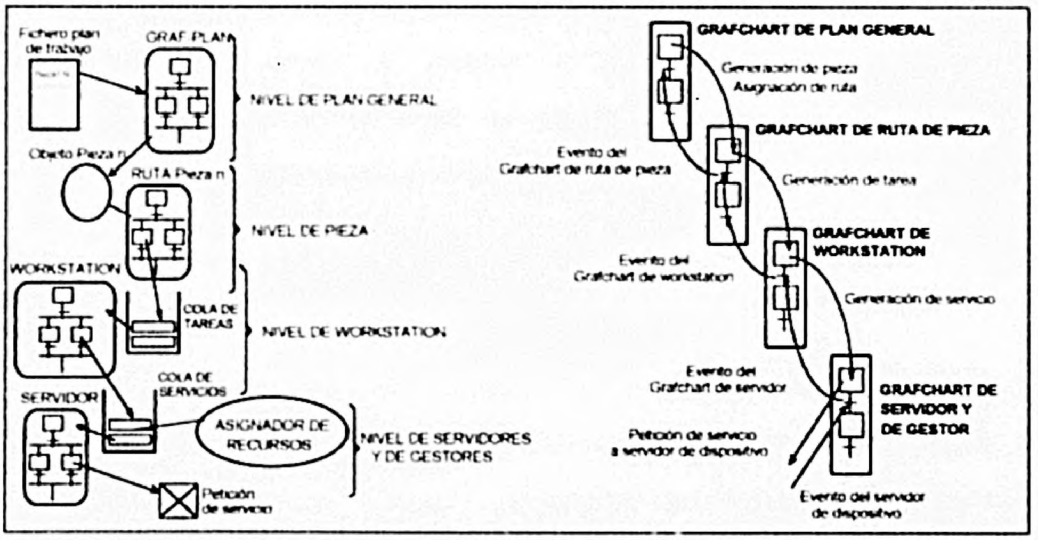


Figura 4.1: Relación entre los diagramas GRAFCET.

4.2.4.2 Proceso de adaptación.

Este proceso se realiza a través de los siguientes pasos:

- **Recuperar del repositorio la base de conocimiento de G2.** Modificarla para que contenga la *definición de las clases de objetos interface y objetos propios de la aplicación*, la declaración de todas las instancias necesarias de dichas clases, y las variables interface de datos entre aplicaciones con su asociación al objeto interface.

- **Recuperar del repositorio el diagrama principal.** Adaptar el *fichero de dispositivos* que indica a la aplicación el conjunto de dispositivos del sistema, sustituir los identificadores del dominio por los particulares, y modificar las siguientes *macros* según las necesidades particulares:

  - *Macro de carga inicial de herramientas y programas.* Desechar aquellas ramas del diagrama que no se adecuen a la forma de realizar la carga inicial en el sistema particular, eliminar todos los procedimientos G2 asociados con dichas ramas, y modificar los procedimientos asociados con ramas utilizadas por más de un dispositivo para que contemple a todos.
  - *Macro de ejecución de un plan.* Establecer la dimensión de las siguientes *listas:* lista de tareas con el identificador de la pieza, código, atada, máquina y fecha, que se recogen del fichero con el plan; la lista de piezas con sus operaciones, que limita el número de piezas que pueden existir simultáneamente en la planta; las listas de operaciones de las distintas máquinas, que deben estar ordenadas cronológicamente.

    Adaptar la *tabla de relación* que asocia cada pieza con su conjunto de operaciones, que permitirá crear la lista de piezas comentada.

- **Recuperar del repositorio el diagrama de ruta de pieza.** Sustituir los identificadores del dominio, modificar las *macros* de paletizado, asignación de transporte, herramientas/programas, carga, mecanizado, descarga y

despaletizado para incluir todas las máquinas de la planta, y por supuesto, realizar las modificaciones a los procedimientos G2 asociados.

- **Recuperar del repositorio los diagramas de workstations.** Replicar los diagramas de workstations de mecanizado (torno y centro de mecanizado) tantas veces como máquinas de este tipo existan en la planta, puesto que normalmente los diagramas de workstations de transporte, paletizado y despaletizado serán únicos.

  En cada caso, preservar aquellas etapas que se correspondan con la forma de realizar las funciones de la workstation en cuestión, eliminar los procedimientos G2 asociados a las ramas desechadas, y sustituir los identificadores del dominio por los particulares.

- **Recuperar del repositorio los diagramas de gestores de recursos.** En ambos casos, eliminar aquellas ramas o etapas que se correspondan con peticiones de servicio a otros módulos no identificadas en la especificación del módulo CBD, así como sus procedimientos G2 asociados.

  Sustituir los identificadores del dominio por los particulares, y replicar aquellas etapas (con sus procedimientos G2 asociados) que solicitan la misma petición de servicio pero para distintos dispositivos (por ejemplo: la modificación de los programas de control numérico para dos tornos o dos centros de mecanizado).

- **Recuperar del repositorio los diagramas de servidores de dispositivos.** Replicar cada tipo de diagrama de servidor del dominio tantas veces como servidores reales de ese tipo existan en la aplicación. Los diagramas obtenidos hay que particularizarlos mediante la eliminación de todas las etapas que se correspondan con peticiones de servicio no identificadas en la especificación del módulo CBD, la supresión de todos los procedimientos G2 asociados a dichas etapas, y la sustitución de los identificadores del dominio por los particulares.

  El componente interface con otros módulos está incluido en los procedimientos G2 para los diagramas de gestores de recursos y servidores de dispositivos. Como en el caso del Scheduler dinámico, es preciso *construir un bridge entre*

GSI y DAE, para lo cual *se dispone del generador de bridges para G2 del bus de mensajes.*

- **Recuperar del repositorio el componente asignador de recursos.** Sustituir los identificadores del dominio, y adaptar las *reglas* que se aplican para gestionar las colas de mensajes de los servidores que atienden servicios de tipo cooperativo.

4.2.4.3 Componentes de prueba.

Las pruebas se realizan mediante un *software de simulación* que se comporta como los restantes módulos en cuanto al envío y recepción de mensajes. Este software permite visualizar las peticiones de servicio que llegan desde el CBD, y generar los eventos y peticiones de servicio hacia el mismo con los parámetros adecuados.

4.2.5 Módulos gestores de recursos.

4.2.5.1 Solución propuesta.

Los gestores de recursos implementados en fases previas no empleaban una base de datos real, sino que simulaban su funcionamiento mediante ficheros y tablas de relaciones.

Los gestores de recursos actuales están desarrollados con el *lenguaje de programación C y la base de datos Access*[36] *[Microsoft, 96]*, se ejecutan *en ordenadores IBM PS 2 con sistema operativo OS 2, y utilizan la plataforma de integración DAE (Distribution Automation Edition) [IBM, 91]* para comunicarse con los restantes módulos.

---

[36] La base de datos de herramientas estará en el almacén general de herramientas, donde pueda ser actualizada con los datos procedentes de la máquina de pre-reglaje y del operario.

Además, *para la construcción del componente gestor de servicios y eventos se ha desarrollado una plantilla* que consta de la sección de declaraciones genéricas de cualquier recurso DAE, y del programa principal. Este patrón se emplea para crear el fichero fuente, puede ser editado con cualquier editor, y no requiere ninguna herramienta adicional para ser modificado.

4.2.5.2 Proceso de adaptación.

Al igual que los demás módulos, este proceso establece las **tareas** que se ejecutan para adaptar los gestores de recursos del dominio a una aplicación particular, una vez definida la correspondiente especificación de mensajería:

- **Recuperar del repositorio el código del componente gestor de servicios y eventos para el tipo de gestor correspondiente.** Eliminar todos aquellas funciones que implementen mensajes no identificados en su especificación, replicar aquellas funciones que solicitan la misma petición de servicio a varios módulos y/o procesan el mismo evento procedente de distintos módulos, replicar también aquellas funciones de peticiones de servicio y eventos que deba realizar el gestor pero para distintos dispositivos, y sustituir los identificadores del dominio por los correspondientes a la aplicación particular.

  Si fuera preciso, adaptar los *ficheros de intercambio* y las *tablas de la base de datos*. También hay que *insertar la información* correspondiente a la aplicación particular en las bases de datos.

  En el caso del gestor de herramientas, hay que adecuar las *tablas de relaciones* que asocian la máquina con la capacidad de su carrusel, y el número de herramientas y sus códigos con el programa de mecanizado (código de pieza y atada) y la máquina.

  Para el gestor de programas de control numérico, hay que adaptar el árbol de directorios y grabar los programas tipo de las máquinas. A partir de estos

programas tipo se obtienen los ficheros (modificados con la información del gestor de herramientas) que se envían a las máquinas de mecanizado.

Este componente incluye la *interface con otros módulos*, que *necesita la librería DAE del bus de mensajes* para realizar la comunicación.

- **Recuperar del repositorio el código del componente driver (común para ambos tipos de gestores).** Si la base de datos no es Access, habrá que construir un nuevo driver o adaptar el existente. En cualquier caso, hay que establecer los identificadores correspondientes a la aplicación particular.

4.2.5.3 Componentes de prueba.

Cada tipo de gestor de recursos dispone de una *interface de usuario con el gestor de servicios y eventos* que permite solicitar cualquier petición de servicio o evento que esté identificado en la especificación de mensajería del dominio para ese tipo de módulo. Dicho software permite comprobar las respuestas (mensajes que emite el componente).

También existe una *interface de usuario con el driver* de la base de datos. Este programa posibilita la solicitud de preguntas a la base de datos, pudiendo observar las respuestas que genera.

4.2.6 Módulos servidores de dispositivos.

4.2.6.1 Solución propuesta.

En este proyecto particular, los servidores del dominio son para máquinas de control numérico, robots, transporte y operario. En relación con los trabajos previos *[Burgos, 94a, 96, 97], [Llorente, 96]* se han reutilizado los drivers de los dispositivos en la construcción de los módulos particulares en la ingeniería de aplicación, desarrollados

los servidores en el mismo *lenguaje de programación C*, y ejecutados *en ordenadores IBM PS 2 con sistema operativo OS 2.*

Además en la solución propuesta en el proyecto piloto:

- Los servidores del dominio se comunican con los demás módulos de control por medio de la *plataforma de integración DAE [IBM, 91]*, y para conectarse con los controladores de planta se ha adoptado la solución RIC; es decir, utilizan *tarjetas de comunicación inteligentes RIC Multiport 2 (Realtime Interface Co-Procesor) [IBM, 89].* Estos elementos garantizan la multitarea necesaria para la implementación de estos módulos.

- Los servidores de dispositivos son *independientes de la solución RIC adoptada* para la comunicación con los controladores de planta. Existe la *librería RIC* que aisla todas las tareas relacionadas con la tarjeta, las cuales se deben realizar con sus propias funciones (RICES – RIC Extended Services) *[IBM, 89].*

- Se utiliza la misma *plantilla DAE anterior para construir los componentes gestor de servicios y eventos, y espía.*

- Se dispone de una *plantilla RIC para el desarrollo del componente núcleo del driver*, que consta de las declaraciones genéricas de cualquier recurso DAE que trabaje en una tarjeta RIC y del programa principal.

En la citada librería RIC están implementadas las siguientes *tareas: preparar la tarjeta para cargar una tarea, descargar una tarea de la tarjeta, inicializar el puerto de comunicaciones, generar las tablas de control para la lectura de mensajes de longitud variable, configurar el puerto de comunicaciones, abrir un puerto, cerrar un puerto liberando los buffers asociados, configurar los parámetros de comunicación para un puerto, enviar datos por la línea de comunicaciones, definir un buffer para los datos de entrada de la tarjeta, y recibir datos de la tarjeta.*

Esta librería dispone de las declaraciones y ficheros "include" necesarios, y ha sido sometida a un *exhaustivo plan de pruebas* que incluye técnicas de caja blanca y negra.

4.2.6.2 Proceso de adaptación.

Conocido el tipo de servidor que se desea implementar y definida su especificación de mensajería, se realizan los siguientes **pasos:**

- **Recuperar del repositorio el código del componente gestor de servicios y eventos para este tipo de servidor.** Eliminar todos aquellas funciones que implementen mensajes no identificados en su especificación, replicar aquellas funciones que solicitan la misma petición de servicio a varios módulos y/o procesan el mismo evento procedente de distintos módulos, y sustituir los identificadores del dominio por los correspondientes a la aplicación particular.

  Eliminar o adaptar los *datos de estado:*

  - Datos de estado *generales* (de todos los servidores del dominio): estado del dispositivo (parado, en ejecución, error, mantenimiento), estado de la comunicación (con o sin transmisión), código/s de error.
  - Datos de estado *del servidor de máquinas de control numérico:* situación del carrusel (código de herramienta en cada posición), situación del palet de entrada o mordaza (fuera, dentro / con, sin pieza), código de la herramienta activa, estado de la puerta (abierta, cerrada).
  - Datos de estado *del servidor de robots:* situación de las garras, garra activa.
  - Datos de estado *del servidor de transporte:* mapa de posiciones de transporte, palet existente en cada posición (código de la pieza o herramienta que transporta).

  Eliminar o adaptar las *tablas de relación:*

  - Tablas *generales:* relación entre los códigos de error transmitidos por el dispositivo y los emitidos en los eventos.
  - Tablas del *servidor de máquinas de control numérico:* relación entre las posiciones de los palets de entrada (o mordaza) y los programas CN que los mueven, entre los códigos de pieza/atada y los programas de CN a ejecutar, entre los códigos de herramienta y sus posiciones en el carrusel, vida

remanente y datos geométricos, entre las posiciones del carrusel y los programas de CN que las mueven a las posiciones de carga/descarga.

- Tablas del *servidor de robots:* relación entre las acciones y la secuencia de programas a ejecutar, entre los códigos de pieza y las garras que las manipulan, entre los códigos de pieza/atada y el programa a ejecutar, entre los códigos de herramientas y las garras que las manipulan.
- Tablas del *servidor de transporte:* relación entre los identificadores de las posiciones y las posiciones físicas del transporte.

Este componente se ejecuta en el procesador del ordenador, e incluye el componente *interface con otros módulos que necesita la librería DAE del bus de mensajes* para poder comunicarse con los restantes módulos.

- **Recuperar del repositorio el código del núcleo del driver** o en su defecto, de la plantilla RIC para construirlo, y adecuar los identificadores a la aplicación particular. Este componente se ejecuta en el procesador de la tarjeta RIC, descargando de este modo al procesador del ordenador y permitiendo así un control de los dispositivos de planta en tiempo real.

- Si es necesario, **recuperar del repositorio el código del componente espía** y modificar los identificadores del dominio. Aunque forma parte del driver, este componente se ejecuta en el procesador del ordenador porque no necesita conexión RS-232 (no se conecta con ningún dispositivo de planta), reduciendo así los posibles problemas de saturación de la capacidad de la memoria RAM de la tarjeta RIC (1 MB).

4.2.6.3 Componentes de prueba.

Para la realización de las pruebas se han construido dos componentes por cada tipo de servidor:

- *Interface de usuario con el gestor de servicios y eventos.* Permite solicitar cualquier petición de servicio o evento que esté implementado en el tipo de

servidor del dominio correspondiente, pudiendo así comprobar sus respuestas (peticiones de servicio y/o eventos que emite el gestor de servicios y eventos).

- *Interface de usuario con el driver* (salvo para el servidor de operario que no dispone de driver). Permite enviar el conjunto de comandos que se le pueden solicitar al controlador de cada tipo de dispositivo, y observar las respuestas que genera (eventos que envía el driver).

4.2.7 Bus de mensajes.

Relacionados con el bus de mensajes, están los siguientes componentes:

- **Protocolo de mensajería**, que define cómo los diferentes módulos se intercambian los mensajes.
- **Plataforma de integración**, utilizada para la implementación del bus.
- **Librerías**, para facilitar la portabilidad de los módulos.
- **Generadores de bridges**, para comunicar el Scheduler dinámico y el CBD con los restantes módulos conectados al bus de mensajes.

Estos componentes se presentan a continuación, indicando cuando sea el caso la contribución realizada.

4.2.7.1 Protocolo de mensajería.

Las peticiones de servicio y los eventos que relacionan los módulos de la arquitectura se traducen en mensajes. El intercambio de dichos mensajes sugiere la necesidad de **definir:**

- Los *tipos básicos de variables* (short, long, byte, char, etc.) *y las variables complejas* (array, struct, etc.) empleadas por las aplicaciones internamente y en el intercambio de mensajes con otros módulos.

- Los *identificadores de las peticiones de servicio.*
- Los *identificadores de los eventos.*
- El *mensaje* está definido como un conjunto de campos, algunos de ellos comunes a todos los mensajes y otros condicionados a las necesidades de la petición de servicio o evento. Estos campos son:
  - *Tipo.* Indica si el mensaje implementa una petición de servicio o un evento.
  - *Identificador.* Identifica la petición de servicio o evento.
  - *Error.* Si el servicio se ha realizado sin problemas, el código de error es cero. En caso contrario, indica el código de error ocurrido.
  - *Origen.* Especifica el nombre del recurso que envía el mensaje.
  - *Destino.* Recoge el nombre del recurso que debe recibir el mensaje.
  - *Prioridad.* Indica la prioridad del mensaje.
  - *Protocolo.* Refleja el tipo de protocolo que se ha especificado para el mensaje entre los que existen en el dominio.
  - *Px.* Reservado para uso de los programadores del sistema.
  - *Longitud.* Contiene la longitud del mensaje que se envía.
  - *Buffer.* Almacena los datos propios de la petición de servicio o evento.

4.2.7.2 Plataforma de integración.

La implementación del bus de mensajes emplea la plataforma de integración *DAE (Distributed Automation Edition) [IBM, 91],* que es un gestor de sistemas en entornos distribuidos.

Las aplicaciones o entidades residentes en los nodos DAE se consideran recursos, y deben ser dados de alta para que los restantes recursos tengan conocimiento de su existencia con objeto de su manipulación.

DAE permite:

- *Establecer una comunicación entre procesos transparente para el usuario.* Esto significa que éste no necesita conocer el modo de conexión de los equipos, el lugar de almacenamiento de los datos o el control de los puertos de comunicación. La aplicación simplemente centra su esfuerzo en la resolución del problema.

- *Realizar el desarrollo de aplicaciones independientes de los dispositivos* que se van a comunicar o controlar. Esta independencia se obtiene por el tratamiento que da a sus recursos. Cada recurso se referencia mediante un nombre único o "alias", que es el identificador empleado en el envío y recepción de los mensajes. Por ello, si se sustituye un dispositivo, sencillamente se elimina del control y ese mismo alias puede ser heredado por un nuevo dispositivo al igual que el programa de usuario existente.

Por otra parte, DAE también está *preparada para trabajar con la tarjeta de comunicaciones RIC Multiport 2 [IBM, 89].* Proporciona un conjunto de programa para su manejo, y la interface de comunicaciones con los procesos que se ejecutan en el procesador de la tarjeta, permitiendo tanto la comunicación entre los procesos que se ejecutan dentro de la RIC, como la comunicación de dichos procesos con los que se ejecutan en la unidad central del equipo *[Burgos, 95].*

4.2.7.3 Librerías.

Para facilitar la portabilidad de los módulos de control a otra plataforma, se ha desarrollado la **librería DAE.** Ésta consta de un conjunto de funciones desarrolladas utilizando las APIs propias de DAE, es decir, se rellenan una serie de campos en las estructuras definidas para tal efecto y se llama a la función DAE correspondiente pasándole estos parámetros. Concretamente, las *funciones* realizan las siguientes tareas: *inicializar un recurso en el entorno DAE, salir de este entorno, leer un mensaje de la cola del recurso, enviar un mensaje a cualquier otro recurso, mandar la orden de*

arrancar un recurso, mandar la orden de parar un recurso, y presentar en pantalla información general o de error.

Con la misma filosofía se ha desarrollado la **librería TCP/IP,** que implementa las *funciones* de: *inicializar socket, enviar un mensaje y recibir un mensaje.*

Ambas librerías disponen de las declaraciones y ficheros "include" necesarios, y han sido sometidas a un *exhaustivo plan de pruebas* que incluye técnicas de caja blanca y de caja negra.

4.2.7.4 Generadores de bridges.

En el marco del proyecto se han desarrollado los **bridges para G2** (GSI/DAE) y **otro para Witness** (GSI/OLE), que se presentan a continuación.

4.2.7.4.1 Generador de bridges para G2.

Un bridge para G2 es un software que permite al Scheduler dinámico y CBD comunicarse con los restantes módulos conectados al bus de mensajes. Dicho software consta de *dos procesos que intercambian información mediante dos sockets:*

- *Bridge UNIX.* Recibe mensajes de módulos con GSI por medio de RPCs, los adapta al protocolo de mensajería del bus de mensajes, y los envía al otro proceso (bridge OS/2) a través del socket de envío. En el otro sentido, extrae la información del mensaje recibido del bridge OS/2, y ejecuta la RPC correspondiente en GSI. Se ejecuta en estaciones RS-6000 de IBM con sistema operativo UNIX.
- *Bridge OS/2.* Intercambia mensajes de recursos DAE con el bridge UNIX. No necesita modificar los mensajes porque él es un recurso DAE más, que se ejecuta en ordenadores IBM PS/2 con sistema operativo OS/2.

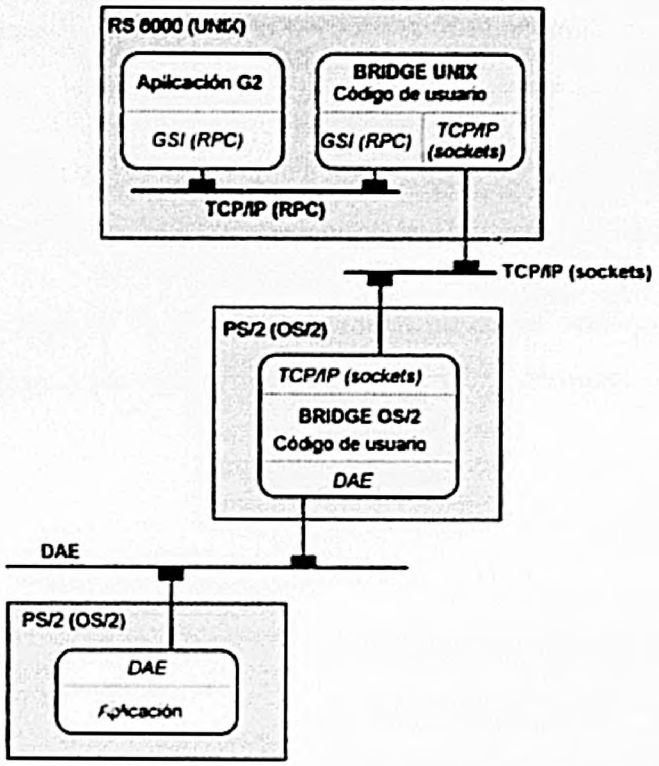


Figura 4.2: Bridge para G2 (GSI – DAE).

Inicialmente, se desarrollaron bridges particulares para cada uno de los módulos, y posteriormente se construyó una herramienta de generación de los mismos *[Burgos, 97]*. Para el presente proyecto piloto se ha ampliado esta herramienta para incluir todos los mensajes identificados en la especificación de mensajería del dominio que se pueden intercambiar con estos módulos.

El *generador de bridges para G2 es una herramienta visual* que permite obtener el código de estos programas de comunicaciones por medio de una serie de menús y ventanas. Se ejecuta en un ordenador *IBM PS/2 con sistema operativo OS/2*, y está implementado con el *lenguaje de programación C* utilizando la *herramienta de desarrollo visual Vispro [HockWare, 95]*. Esta herramienta permite la realización de una interface que se ajusta a los estándares propuestos en el entorno gráfico Presentation Manager[37] de OS/2.

---

[37] Vispro y Presentation Manager son a OS/2 lo que Motif y XWindows son a Unix.

Para la **generación de un nuevo bridge** se realizan los siguientes pasos:

- *Definir los elementos que van a ser comunicados mediante el bridge.*
- *Seleccionar los mensajes que van a ser transmitidos a través del bridge.*
- *Generar los ficheros fuente que componen el bridge.* Este proceso consiste básicamente en la combinación de:
  - Ficheros de datos, denominados ficheros del sistema, que contienen la información sobre los proyectos, bridges, mensajes y parámetros.
  - Plantillas, que son patrones del contenido de los ficheros fuentes resultantes.
- *Compilar los ficheros fuente en los sistemas destino (UNIX u OS/2).*

Las pruebas se realizan mediante *dos software de simulación* que permiten el envío y la recepción de mensajes en ambos extremos. De este modo, con el envío de mensajes desde el componente GSI, se comprueba que el bridge UNIX transmite y el bridge OS/2 recibe, mientras que con el envío de mensajes desde el componente DAE, se comprueba que el bridge OS/2 transmite y el bridge UNIX recibe.

4.2.7.4.2 Generador de bridges para Witness.

El generador anterior ha sido ampliado para construir también *bridges para Witness*. La parte del *bridge Windows* está desarrollada con *Microsoft Visual C++*[38] con algunas llamadas a funciones en C, mientras que la parte de generación del *bridge UNIX* se ha reutilizado. Los procesos en UNIX se comunican con Windows mediante *Winsockets [Quinn, 96] a través del protocolo de comunicaciones TCP/IP.*

El *bridge Windows tiene que transformar los mensajes* que llegan a la aplicación WITNESS al formato que ésta utiliza para almacenar la información, y de este formato al formato de mensaje del bridge UNIX para los que se envían.

---

[38] Herramienta de desarrollo visual orientada a objetos que incluye las librerías de Winsockets y la comunicación con otras aplicaciones mediante OLE Automation.

Para componentes de pruebas se ha reutilizado el de GSI y se ha desarrollado uno análogo para la parte Windows.

4.2.8 Sistemas de comunicaciones industriales.

4.2.8.1 Conexión entre los módulos.

Como se puede observar en la figura 4.3, los servidores de dispositivos y los gestores de recursos están conectados a una red *Token Ring, con la plataforma de integración DAE [IBM, 91]* como medio de intercomunicación entre los mismos. Los restantes módulos están conectados a la citada red *Token Ring o a una red Ethernet, y utilizan el protocolo TCP/IP* como medio de intercomunicación. La interconexión entre las redes Token Ring y Ethernet se realiza mediante un Router *[Gómez, 94]*.

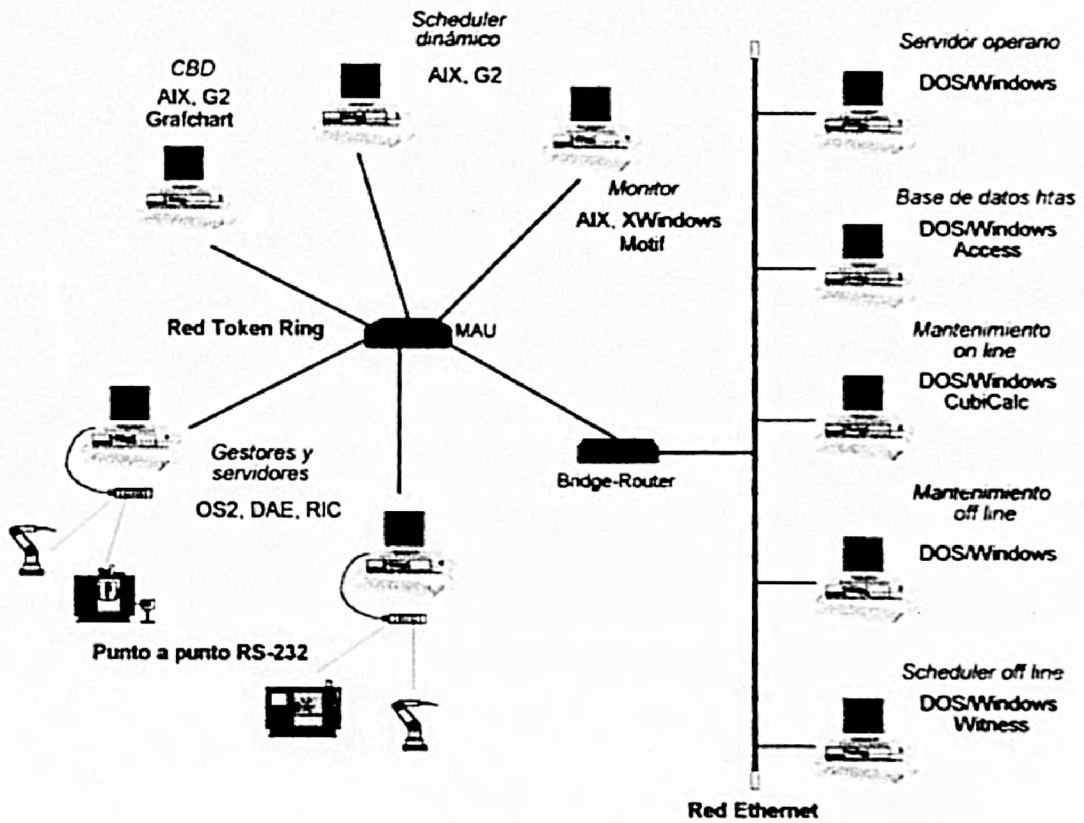


Figura 4.3: Sistemas de comunicaciones.

4.2.8.2 Conexión con los dispositivos de planta.

La conexión con los dispositivos de control de planta es vía serie RS-232 (ver figura 4.3), por medio de tarjetas RIC Multiport/2 *[IBM, 89]*. Estas tarjetas disponen de ocho puertos serie que se conectan *punto a punto con las interfaces RS-232 de los dispositivos.*

4.2.9 Repositorio.

En los trabajos previos la gestión de los productos se ha realizado mediante una *estructura de directorios.* En esta estructura se disponía de un directorio general con los productos de la planta, y de un directorio para cada módulo con distintos subdirectorios para cada versión.

No se realizaba ninguna distinción entre los componentes de los módulos, ni entre los productos generados para cada módulo en las distintas etapas del proceso de desarrollo, y tampoco había información sobre el contenido de los directorios.

4.2.9.1 Solución propuesta.

Para el presente proyecto, el repositorio se ha construido utilizando la *herramienta B-Kin Links [B-Kin, 96].* Esta herramienta *emplea la base de datos orientada a objetos ObjectStore [Object, 97],* para realizar una gestión efectiva de los elementos que almacena. La *interface de usuario está implementada con Visual C++,* y se ejecuta en un *PC con sistema operativo MS-DOS y entorno Windows.*

Este desarrollo ha sido sometido a un *exhaustivo plan de pruebas con carga de documentos reales (tipo Word y Power Point), y pruebas de resistencia y de seguridad, verificando simultáneamente el correcto funcionamiento de la interface de usuario* que permite utilizar las distintas opciones de la aplicación.

4.2.9.2 Procesos para la ingeniería de dominio.

Todos los productos generados en la ingeniería de dominio se han insertado en el repositorio, de acuerdo con los cinco niveles de clasificación y los items que se relacionan con los anteriores mediante las correspondientes relaciones de librería.

Una vez creado y abierto el repositorio, el proceso de inserción de elementos consta de los siguientes pasos: *creación de los tipos de ítems, definición de las relaciones, definición de los ítems concretos, y asignación de valores a los atributos de los ítems.*

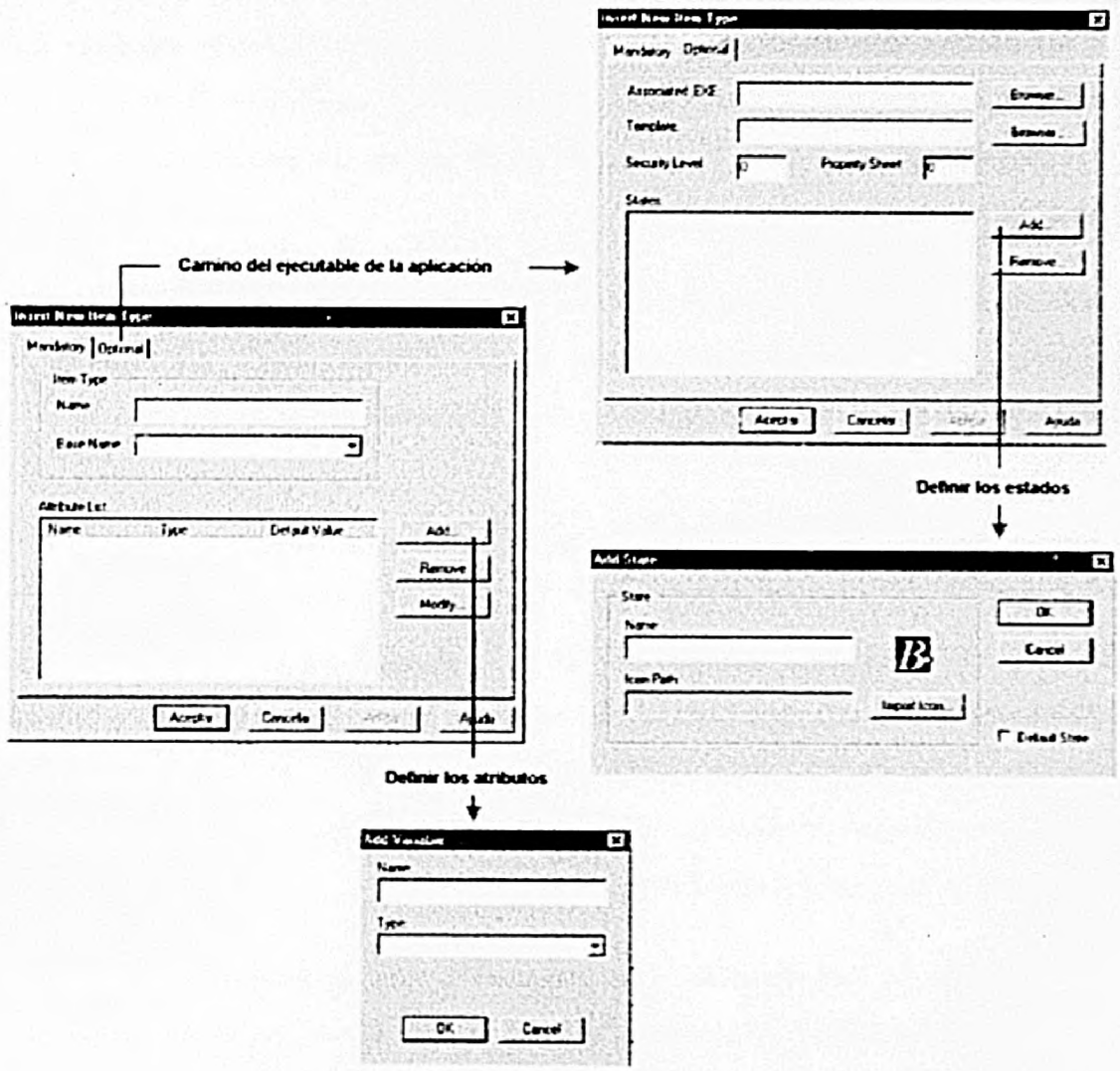


Figura 4.4: Creación de los tipos de items.

El **proceso de borrado** de elementos requiere la *selección del ítem, y la ejecución de las siguientes opciones del menú: Ítem, Delete Ítem.*

El **proceso de edición y modificación** de elementos consiste en la *selección del ítem, y la ejecución de las siguientes opciones del menú: Ítem, File, Edit.*

El **control de versiones** se garantiza mediante el ítem de históricos que existe en cada nivel de clasificación, y el ítem **índice en cada nivel** proporciona información sobre los distintos elementos que se encuentran en dicho nivel.

4.2.9.3 Procesos para la ingeniería de aplicación.

La **consulta del repositorio** se realiza mediante una *interface de usuario* similar al explorador de Windows. A la hora de realizar operaciones, se requiere simplemente el ratón y tener los permisos oportunos.

El *índice de cada nivel ayuda en la búsqueda* de los elementos. No obstante, cada elemento precisa una *buena documentación de reutilización (adaptación),* que proporcione una perfecta descripción de las posibilidades del elemento y las acciones necesarias para poder utilizarlo en otras aplicaciones.

4.3 Ingeniería de aplicación.

La *construcción del software de control para el sistema de fabricación* sito en el taller del Dpto. de Ingeniería Mecánica de la Escuela Técnica Superior de Ingenieros, se ha realizado *a partir de los productos generados en las distintas etapas del proceso de ingeniería de dominio y de acuerdo con los procesos descritos en las tareas de "Soporte a la ingeniería de aplicación" correspondientes.*

La planta piloto en la que se ha implementado la metodología propuesta consta de los siguientes dispositivos: robot Fanuc S-700, robot Fanuc S-10, robot Asea IRB6, centro de mecanizado Kondia B-500 con CN Fidia, centro de fresado Kondia K-76 con CN Fagor 8010-M, centro de torneado TBI CMZ con CN Fagor 8050-F y sistema de transporte TS-2 controlado por un PLC de Siemens.

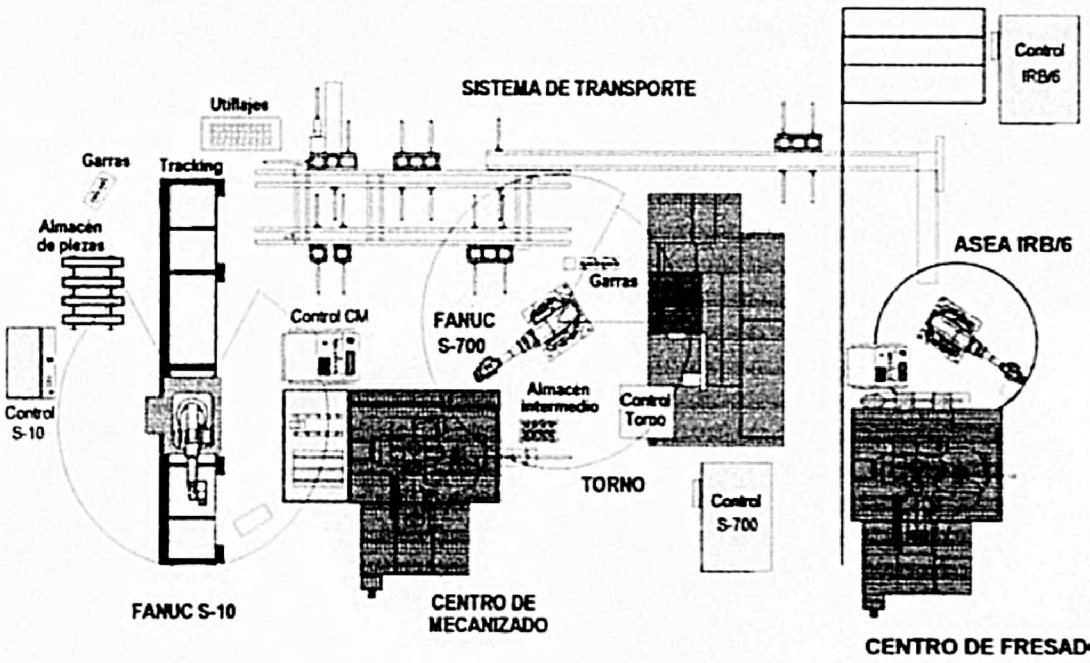


Figura 4.5: Layout de la planta de fabricación.

La **gestión del proyecto** se ha realizado siguiendo las directrices de la ingeniería de dominio: la planificación tiene en cuenta la reutilización de componentes en todas las etapas, y los procesos de evaluación de la calidad y las tareas de gestión de la configuración son los estipulados en la ingeniería de dominio.

El sistema de control final y todos los productos asociados son el resultado de las actividades de **análisis, diseño e implementación del proyecto,** que se describen en las secciones subsiguientes.

4.3.1 Análisis de la aplicación.

Las tareas que se realizan en esta etapa son: **establecer la operativa y las especificaciones del nuevo sistema, siguiendo los procesos descritos en la ingeniería de dominio.**

4.3.1.1 Operativa del sistema.

La operativa del sistema se describe como *un subconjunto de las tareas identificadas para el dominio,* e indican cómo se va a llevar a cabo la manipulación de piezas, de herramientas y de programas, las operaciones generales, de mantenimiento y demás acciones.

De acuerdo con el proceso establecido en la ingeniería de dominio para determinar la operativa del sistema, *la información se organiza en los siguientes términos: almacenes, piezas, herramientas, operaciones de paletizado, máquinas, sistemas de manipulación, y operaciones de mantenimiento.*

A continuación se presenta la información más relevante correspondiente al presente proyecto de ingeniería de aplicación y se identifica el subconjunto de tareas del dominio para el mismo.

4.3.1.1.1 Información de los almacenes.

Existen almacenes de herramientas, piezas, palets, utillajes y garras, distinguiendo a su vez almacenes generales y locales, tal y como se describe a continuación.

1. **Almacenes de herramientas:**

   - *Almacén general de herramientas.* Almacén manual que guarda las herramientas del sistema, excepto aquellas que se encuentran en producción, en las máquinas de prerreglaje o presetting, en reparación o en montaje.

   - *Almacén intermedio de herramientas.* Almacén manual con capacidad para 18 herramientas, que está situado junto al Kondia B500 para darle servicio en las operaciones de vaciado y llenado del carrusel.

   - *Carruseles.* Considerados almacenes por el sistema de gestión de herramientas, su control lo realizan las propias máquinas automáticamente.

2. **Almacenes de piezas:**

   - *Almacén general de materia prima.* Almacén manual que tiene la materia prima de las piezas que pueden fabricarse.

   - *Almacén general de entrada al sistema.* Almacén con piezas para introducir en el sistema. Se abastece de forma manual.

   - *Almacenes locales.* Almacén local junto a Kondia B500 con piezas que sólo requieren operaciones en esa máquina. Su abastecimiento es manual.

3. **Almacenes de palets:**

   - *Almacén general de palets.* Almacén con palets montados y preparados para el transporte. El montaje y el traslado a los almacenes son manuales.

   - *Almacenes locales de palets.* Existe un almacén situado en el sistema de transporte para palets que entran al sistema de forma automática. El otro almacén está en las proximidades de la entrada de palets de forma manual. En ambos casos, el abastecimiento es manual.

4. **Almacenes de utillajes:**

   - *Almacén general de utillajes.* Almacén manual que guarda utillajes.

- *Almacén local de utillajes.* Almacén manual que usa el robot y el operario para realizar las operaciones de paletizado.

5. Almacenes de garras:

- *Almacén general de garras.* Almacén manual con garras.
- *Almacenes locales de garras.* Almacenes abastecidos de forma manual, que están situados junto a los dos robots con cambio automático de garras.

4.3.1.1.2 Información de las piezas.

Existen *12 familias de piezas definidas con un total de 74 piezas.* Para el proyecto piloto se han *elegido tres tipos sencillos*, que demuestran las características genéricas de flexibilidad del sistema sin prolongar los tiempos de mecanizado. Estas piezas son las siguientes:

- **Cenicero.** Consta de dos piezas:
  - *Base del cenicero.* Su código Opitz es 130000, y se mecaniza en *tres atadas:* tres atadas en el torno, o bien dos atadas en el torno y una en el centro de mecanizado o fresado.
  - *Tapa del cenicero.* Su código Opitz es 191600, y se mecaniza en *dos atadas* en el torno.
- **Portalápices.** Su código Opitz es 805000, y se realiza en *tres atadas* en el centro de mecanizado o fresado. La secuencia de taladros de la cara superior y el nombre de la cara frontal se pueden personalizar.
- **Portanotas.** Su código Opitz es 854310, y se fabrica en *una única atada* en el centro de mecanizado o fresado. El nombre de la cara superior se puede personalizar.

El paletizado, entrada y salida del sistema son independientes de la pieza y la ruta de fabricación. Normalmente, el *paletizado es automático* (ver información de paletizado),

la *entrada es en palet* desde la posición de paletizado, y la *salida es en palet* hasta la rampa de salida. No se realiza paletizado cuando el portalápices o el portanotas se mecaniza únicamente en el Kondia B500, porque un robot se encarga de cargar y descargar la pieza de forma directa.

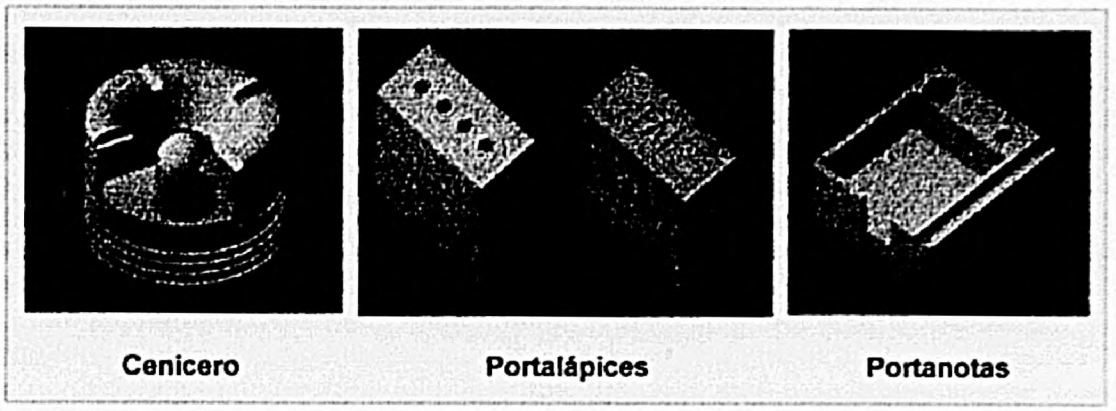


Figura 4.6: Piezas fabricadas en el proyecto piloto.

4.3.1.1.3 Información de las herramientas.

Antes de la ejecución de un plan de trabajo, se debe *comprobar la disponibilidad de las herramientas* mediante accesos a una *base de datos* que almacena la información de todas las herramientas del sistema.

Independientemente de la máquina destino de las herramientas, la *entrada y salida de las mismas se realiza de forma manual sin necesidad de paletizado*, a pesar de que la carga sea automática desde el almacén intermedio para la máquina Kondia B500.

Observaciones. Las *herramientas están codificadas*, y aquellas con carga automática llevan incorporado un chip para poder ser identificadas de forma automática mediante el sistema CIS-ROIN-150-0. Además, la *máquina de pre-reglaje MICROSET* mide el desgaste que sufre la herramienta en el mecanizado, proporcionando datos actualizados de las dimensiones de las herramientas que se guardan en la base de datos.

4.3.1.1.4 Información de las operaciones de paletizado.

Las *operaciones de paletizado se realizan únicamente con piezas*, distinguiendo dos modos:

- **Paletizado automático.** El robot *Fanuc S-10 es el responsable* de colocar los utillajes en el palet vacío, y de situar la pieza en bruto sobre el mismo. Las garras empleadas en cada caso se recogen en la tabla 3.1 (información de los sistemas de manipulación).

- **Paletizado manual.** El *operario es el encargado* de colocar los utillajes y la pieza en el palet vacío, así como de colocar el propio palet lleno en el sistema de transporte. Este modo de paletizado se contempla como excepcional, porque se emplea en casos de avería en el sistema automático o para optimizar tiempos en casos específicos (por ejemplo, en el arranque del sistema).

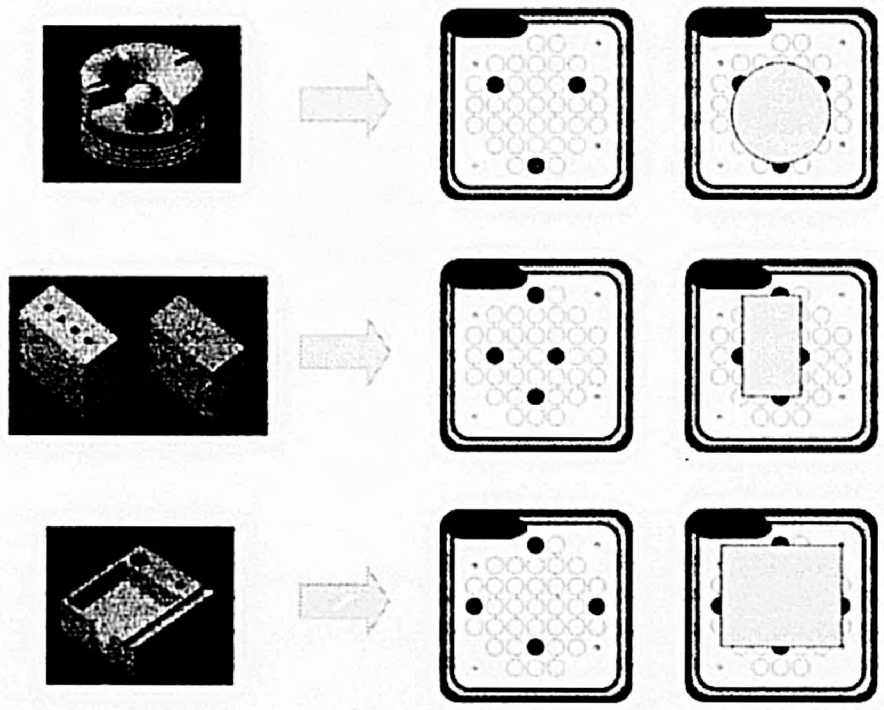


Figura 4.7: Colocación de los utillajes y las piezas en el palet.

Ambos modos de paletizado *emplean el mismo tipo de palet y utillaje*, siendo también independientes de la pieza. El palet consta del portapiezas, el soporte de datos móvil

(información sobre la pieza grabada electrónicamente), y el soporte para los utillajes y la pieza. Este último dispone de 36 agujeros para colocar los utillajes, cuya distribución determina la pieza que se va a fijar para su posterior transporte.

4.3.1.1.5 Información de las máquinas.

Las máquinas de mecanizado son las siguientes:

- **CMZ TBI-450:**
  - *Identificador.* DTO1.
  - *Tipo de máquina.* Centro de torneado.
  - *Descripción.* Torno equipado con eje C, es decir, con control sobre el cabezal. Dicho cabezal tiene instalado un plato que permite el mecanizado de diferentes diámetros de piezas. También dispone de herramientas motorizadas que le permiten hacer funciones de fresado.
  - *Protocolo.* Propio del control numérico FAGOR 8050F (NC1).
  - *Capacidad de la torreta.* 12 plaquitas.
  - *Modo de carga descarga de herramientas.* Manual.
  - *Modo de carga descarga de los programas CN.* Automático.
  - *Modo de carga de los correctores.* Automático.
  - *Operaciones de mecanizado.* Tres atadas de la base del cenicero y dos atadas de la tapa del cenicero. Ninguna de estas operaciones necesita cambio de plato, y para todas ellas se realiza la carga/descarga desde palet.

  Observaciones. En el torno está instalado el dispositivo MONTRONIX TS200 Tool Monitor de Kennametal, que permite conocer los esfuerzos sobre la herramienta en las tres direcciones principales durante el corte.

- **KONDIA B-500.**
  - *Identificador.* DCM1.
  - *Tipo de máquina.* Centro de mecanizado.

- *Descripción.* Dispone de 3 ejes, cambiador automático de herramientas, y además de las funciones propias de los controles normales, posee funciones de palpado y copiado.
- *Protocolo.* Propio del control numérico FIDIA COPYMILL (NC2).
- *Capacidad del carrusel.* 18 herramientas.
- *Modo de carga/descarga de herramientas.* Automático desde el almacén intermedio.
- *Modo de carga descarga de los programas CN.* Automático.
- *Modo de carga de los correctores.* Automático.
- *Operaciones de mecanizado.* Una atada de la base del cenicero, tres atadas del portalápices y una atada del portanotas. Según la pieza que se vaya a mecanizar, se usa la mordaza para piezas cilíndricas o la correspondiente para piezas prismáticas. La carga/descarga se realiza desde palet, salvo carga/descarga directa cuando se mecanizan todas las atadas del portalápices o del portanotas.

Observaciones. Este centro de mecanizado está equipado con un lector de chips de herramientas CIS-ROIN-150-0 para leer el código de las mismas automáticamente, y de un manipulador para cargar/descargar herramientas del carrusel.

- **KONDIA K-76.**
  - *Identificador.* DCM2.
  - *Tipo de máquina.* Centro de fresado.
  - *Descripción.* Tiene 3 ejes, y realiza operaciones de fresado y taladrado.
  - *Protocolo.* Propio del control numérico FAGOR 8010M (NC3).
  - *Capacidad del carrusel.* 24 herramientas.
  - *Modo de carga descarga de herramientas.* Manual.
  - *Modo de carga descarga de los programas CN.* Automático.
  - *Modo de carga de los correctores.* Automático.

- *Operaciones de mecanizado.* Una atada de la base del cenicero, tres atadas del portalápices y una atada del portanotas. Ninguna de estas operaciones necesita cambio de mordaza, y para todas ellas se realiza la carga/descarga desde palet.

4.3.1.1.6 Información de los sistemas de manipulación.

La planta piloto dispone de un sistema de transporte de palets y tres robots:

- **Sistema de transporte TS-2.**
  - *Identificador.* TS2.
  - *Tipo de sistema.* Linea de transporte de palets flexible y modular.
  - *Descripción.* El sistema de circulación principal está formado por cuatro tramos de cinta transportadora, dispuestos en forma de rectángulo, con velocidad y sentido de giro único y constante. El tramo longitudinal hasta el elevador y de éste hasta el centro de fresado son bidireccionales. También existen una serie de derivaciones que permiten la detención de los palets.
  - *Protocolo.* Propio del PLC Simatic S5-115U de Siemens (PLC1).
  - *Modo de carga descarga de programas.* Residentes.
  - *Posiciones de acceso.* Están definidas 13 posiciones (PTR0 - PTR12): PTR0 es la pila de palets, PTR1 es la posición de paletizado, PTR2 es la entrada al sistema de transporte, PTR3/PTR4 son entradas manuales, PTR5/PTR6 son entradas manuales y buffers de espera para el centro de fresado, PTR7 es el acceso al centro de fresado, PTR8/PTR9 son de acceso al torno, PTR10/PTR11 son de acceso al centro de mecanizado, PTR12 es la salida del sistema de transporte (tobogán de salida).
  - *Operaciones de manipulación.* Transporte de piezas sobre palets desde cualquier origen a cualquier destino.

  Observaciones. Los palets disponen de un módulo de identificación y almacenamiento de datos (ID 80), que puede ser leido o escrito por las estaciones de lectura/escritura (SLS). El PLC gobierna el flujo de palets a través

del sistema de transporte, pero las unidades SLS permiten comprobar que el recorrido es el correcto.

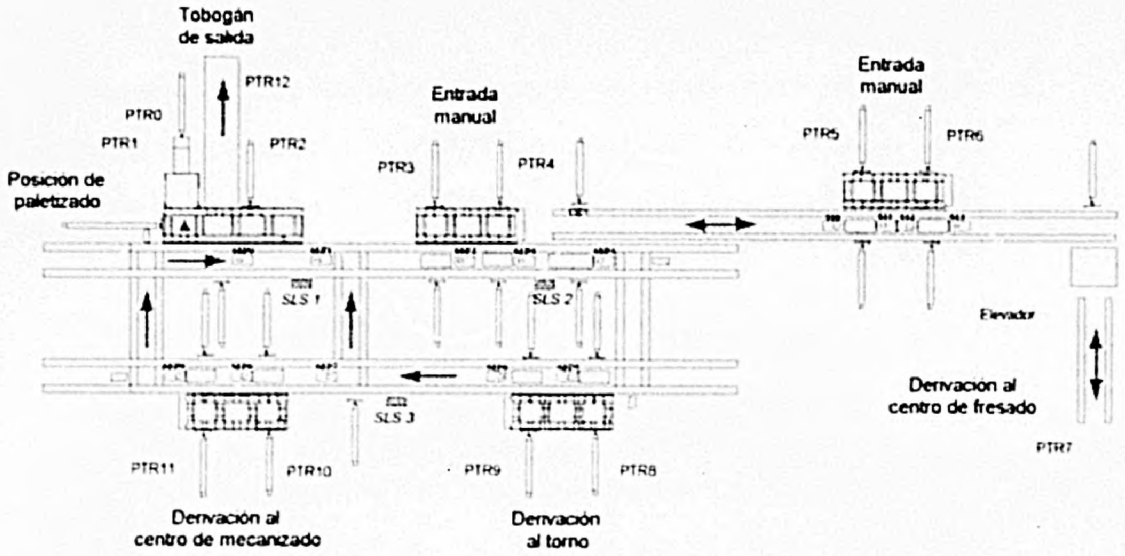


Figura 4.8: Posiciones del sistema de transporte.

- **Fanuc S-700.**
  - *Identificador.* DR1.
  - *Tipo de sistema.* Robot.
  - *Descripción.* Robot de 6 ejes con capacidad para mover piezas de 30 Kg, tiene un alcance máximo de 1616'2 mm, y está equipado con sistema de cambio de garras (CR80 de Röhm).
  - *Protocolo.* Propio del RC Fanuc S-700 (RC1).
  - *Modo de carga descarga de programas.* Residentes.
  - *Posiciones de acceso.* Torno, derivaciones al torno, carrusel del centro de mecanizado, lector de herramientas, almacén intermedio de herramientas, almacén local de garras, mesa auxiliar de volteo de pieza, y reposo.
  - *Operaciones de manipulación.* Carga/descarga de piezas en el torno, y carga/descarga de herramientas en el centro de mecanizado. Las garras empleadas en cada caso se recogen en la tabla 4.2.

- **Fanuc S-10.**
  - *Identificador.* DR2.
  - *Tipo de sistema.* Robot.
  - *Descripción.* Robot de 6 ejes capaz para manipular piezas de hasta 10 Kg, tiene un alcance de 1529 mm, está equipado con sistema de cambio de garras (CR80 de Röhm), y debe controlar un séptimo eje asociado con su movimiento longitudinal.
  - *Protocolo.* Propio del RC Fanuc S-10 (RC2).
  - *Modo de carga descarga de programas.* Residentes.
  - *Posiciones de acceso.* Posición de paletizado, almacén local de utillajes, almacén general de entrada de piezas al sistema, derivaciones al centro de mecanizado, centro de mecanizado, almacén local de garras, mesa auxiliar de volteo de pieza, y reposo.
  - *Operaciones de manipulación.* Paletizado automático de piezas, y carga/descarga de piezas en el centro de mecanizado. Las garras empleadas en cada caso se recogen en la tabla 4.2.

- **Asea IRB6.**
  - *Identificador.* DR3.
  - *Tipo de sistema.* Robot.
  - *Descripción.* Robot de 5 ejes con capacidad para manipular piezas de 6 Kg, y tiene un alcance máxima de 959 mm.
  - *Protocolo.* Propio del Asea IRB6 (RC3).
  - *Modo de carga descarga de programas.* Residentes.
  - *Posiciones de acceso.* Centro de fresado, derivaciones al centro de fresado, mesa auxiliar de volteo de pieza, y reposo.
  - *Operaciones de manipulación.* Carga/descarga de piezas en el centro de fresado. Las garras empleadas en cada caso se recogen en la tabla 4.2.

Elemento	*Fanuc S-700*	*Fanuc S-10*	*Asea IRB6*
Utillajes		G2DR2	
Herramientas	G2DR1		
Base del cenicero	G1DR1	G1DR2	
Tapa del cenicero	G1DR1	G1DR2	
Portalápices		G1DR2	G1DR3
Portanotas		G1DR2	

Tabla 4.2: Garras necesarias en las operaciones de manipulación.

4.3.1.1.7 Información de las operaciones de mantenimiento.

Las operaciones básicas de mantenimiento consisten en la revisión de los dispositivos, almacenes, y posiciones del sistema de transporte. Sin embargo, también se solicitará al operario la revisión de cualquier componente de un dispositivo (batería, programa, frenos, circuito hidráulico, válvulas, etc.). Estas solicitudes se corresponden con las acciones correctoras de los FMEAs de la planta.

4.3.1.1.8 Identificación del subconjunto de tareas del dominio.

Con la información organizada de este modo, se pueden establecer perfectamente aquellas *tareas del conjunto identificado en el proceso de ingeniería de dominio que hay que llevar a cabo en la presente aplicación.*

En la tabla 4.3 se presentan las tareas del dominio, distinguiendo aquellas que no se realizan en esta aplicación (letra común), aquellas que están presentes (letra azul), y en este último caso, si cabe, los distintos casos particulares (letra azul cursiva).

Tareas generales
Ejecución de un plan de trabajo.
Replanificación.
Reserva de herramientas.
Transporte de un palet (pieza o herramienta): *Transporte de palet de pieza.*
Tareas de manipulación de herramientas
Paletizado/despaletizado automático.
Paletizado/despaletizado manual.
Carga/descarga desde palet.
Carga/descarga desde el almacén intermedio: *para el centro de mecanizado (DCM1).*
Carga/descarga manual: *para el torno (DTO1) y el centro de fresado (DCM2).*
Tareas de manipulación de piezas
Paletizado automático: *para cualquier pieza con cualquier destino.*
Paletizado/despaletizado manual: *para cualquier pieza con cualquier destino.*
Despaletizado automático.
Carga/descarga desde palet: *para DTO1, DCM1 y DCM2.*
Carga/descarga directa: *para DCM1.*
Carga/descarga manual.
Tareas de manipulación de programas
Modificación de los programas CN: *para DTO1, DCM1 y DCM2.*
Carga inicial de programas CN y de correctores: *para DTO1, DCM1 y DCM2.*
Mecanizado de una atada: *para DTO1, DCM1 y DCM2.*
Carga inicial de los programas de robots.
Carga posterior de los programas de robots.
Tareas de mantenimiento y otras
Operaciones de mantenimiento.
Solicitud de estado: *para todos los dispositivos.*
Solicitud de fin de proceso: *para todos los módulos.*

Tabla 4.3: Subconjunto de tareas del dominio.

4.3.1.2 Requisitos funcionales del sistema.

Toda la información recogida en la operativa del sistema más los datos de modelado de la planta, las especificaciones técnicas y el layout, *han permitido rellenar la tabla del dominio (tabla 4.4) para identificar los requisitos funcionales* del sistema particular.

¿Se desea realizar			
adaptación del plan de producción a la situación real de la planta?	☐ No	☑ Si	☐ .....
coordinación general de los flujos?	☐ No	☑ Si	☐ .....
gestión de piezas?	☐ No	☑ Si	☐ .....
gestión de herramientas?	☐ No	☑ Si	☐ .....
gestión de palets?	☐ No	☑ Si	☐ .....
gestión de programas?	☑ CN	☐ RC	☐ .....
gestión de los sistemas de transporte?	☐ No	☑ Robot	☑ S.palets
monitorización del sistema?	☐ No	☑ Si	☐ .....
modo de funcionamiento?	☑ Automático		☑ Semi-Aut.
operaciones de mantenimiento?	☐ No	☑ Si	☐ .....
tareas de diagnóstico?	☐ No	☑ Si	☐ .....
control del proceso estadístico?	☐ No	☑ Si	☐ .....

Tabla 4.4: Requisitos funcionales del proyecto piloto.

Con la *aprobación de la especificación de requisitos*, queda concluida la fase de análisis y se inicia la de diseño.

4.3.2 Diseño de la aplicación.

Las **tareas** que se realizan en esta etapa son: **adaptar la arquitectura genérica del dominio** a la instancia concreta de la familia de productos, **detallar las relaciones existentes entre los módulos** identificados a partir de los correspondientes MFDs generados en la ingeniería de dominio, y **concretar las especificaciones de mensajería de los distintos módulos.** Estas tareas se describen en los apartados subsiguientes.

4.3.2.1 Arquitectura específica del nuevo producto.

De acuerdo con las directrices marcadas en la ingeniería de dominio, el primer paso en la identificación de los módulos, que componen la arquitectura de la presente aplicación, se realiza *por medio de los enlaces explícitos existentes entre la especificación y la arquitectura de dominio* (tabla 3.8).

Esta primera versión *se refina según los criterios del dominio para la asignación de gestores de recursos y servidores de dispositivos.* Conocida la arquitectura, *se establecen las relaciones entre los módulos mediante los correspondientes MFDs.*

4.3.2.1.1 Identificación inicial de los módulos.

La *traslación de los requisitos funcionales del presente proyecto* a la tabla citada revela la necesidad de los siguientes módulos: **Monitor, Mantenimiento on-line, Scheduler dinámico, CBD, gestores de recursos, servidores de dispositivos y bus de mensajes,** tal y como se muestra en la tabla 4.5. No obstante, se desconoce el número y tipo de los módulos gestores de recursos y servidores de dispositivos.

Requisitos funcionales	*Módulo*
Monitorización del sistema (incluye interface de usuario), modo de funcionamiento automático y semi-automático, gestión de la información y control del proceso estadístico	Monitor
Operaciones de mantenimiento, tareas de diagnóstico, gestión de la información y control del proceso estadístico	Mantenimiento on-line
Adaptación del plan de producción a la situación real de la planta, gestión de la información y control del proceso estadístico	Scheduler dinámico
Coordinación general de los flujos, gestión de la información y control del proceso estadístico	Control básico Dispatcher
Gestión de palets, de programas CN, de herramientas, de robots, del sistema de transporte de palets, y de la información	Gestores de recursos
Gestión de piezas, de palets, de programas CN, de herramientas, de robots, del sistema de transporte de palets, y de la información	Servidores de dispositivos
Integración de los módulos	Bus de mensajes

Tabla 4.5: Vinculación entre los requisitos del proyecto y la arquitectura del dominio.

4.3.2.1.2 Refinamiento de la identificación de los módulos.

La *aplicación de los criterios del dominio* para discernir el número y tipo de gestores de recursos y servidores de dispositivos, *añade los siguientes módulos:*

- **Dos gestores de recursos: un gestor de herramientas y un gestor de programas de control numérico.** Las herramientas y los programas de control numérico son los recursos del sistema de más difícil gestión, porque son

recursos compartidos por tres máquinas de mecanizado, y por lo tanto, su gestión excede el ámbito de los servidores de dispositivos.

- **Ocho servidores de dispositivos.** Cada máquina de mecanizado, cada robot, el sistema de transporte y el operario precisan su propio servidor de dispositivo, de modo que se han identificado: **tres servidores para máquinas de control numérico, tres servidores de robots, un servidor de transporte y un servidor de operario.**

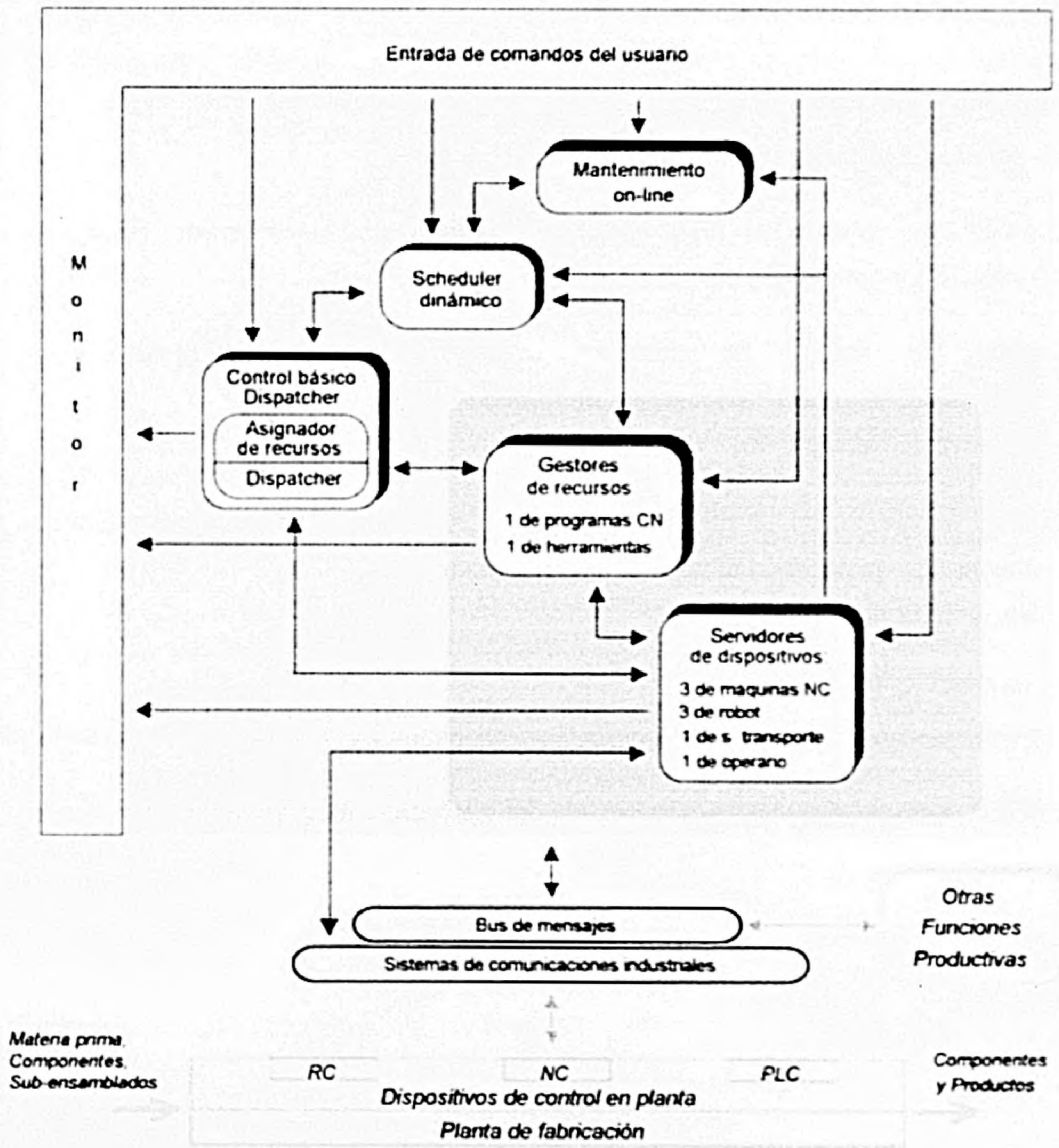


Figura 4.9: Arquitectura para el proyecto piloto.

4.3.2.2 Relaciones entre los módulos de la arquitectura.

Las relaciones existentes entre los módulos identificados en esta arquitectura particular *se recogen en los MFDs de las tareas indicadas en la operativa del sistema* (subconjunto de las tareas del dominio). Estos MFDs se han construido a partir de los correspondientes del dominio: *para cada tarea concreta, se ha recuperado del repositorio su MFD del dominio, y se ha adaptado a este proyecto concreto.*

En la mayoria de los casos, *la adaptación ha consistido en sustituir los identificadores* de los servidores utilizados en los MFDs del dominio por los particulares del proyecto piloto, ya que los mensajes y la secuencia de los mismos no han variado.

Identificadores del dominio	*Identificadores de la aplicación*
MONITOR (Monitor)	MONITOR
MNTO (Mantenimiento on-line)	MNTO
SCH (Scheduler dinámico)	SCH
CBD (Control básico Dispatcher)	CBD
GHTAS (Gestor de herramientas)	GHTAS
GDNC (Gestor de programas CN)	GDNC
SNC (Servidor de máquina CN)	SNC1 (Servidor del DTO1) SNC2 (Servidor del DCM1) SNC3 (Servidor del DCM2)
SRC (Servidor de robot)	SRC1 (Servidor del DR1) SRC2 (Servidor del DR2) SRC3 (Servidor del DR3)
SPLC (Servidor de transporte)	SPLC (Servidor del TS2)
OPLAN (Servidor de operario)	OPLAN

Tabla 4.6: Identificadores del dominio y de la aplicación particular.

Las tareas que presentan distintos casos particulares, tienen cada uno de ellos un MFD, aunque se han obtenido todos ellos del mismo MFD del dominio. Estas tareas son: carga/descarga manual de herramientas para DTO1 y DCM2, y carga/descarga pieza desde palet, modificación de programas CN, carga inicial de programas CN con correctores, y mecanizado de una atada para DTO1, DCM1 y DCM2.

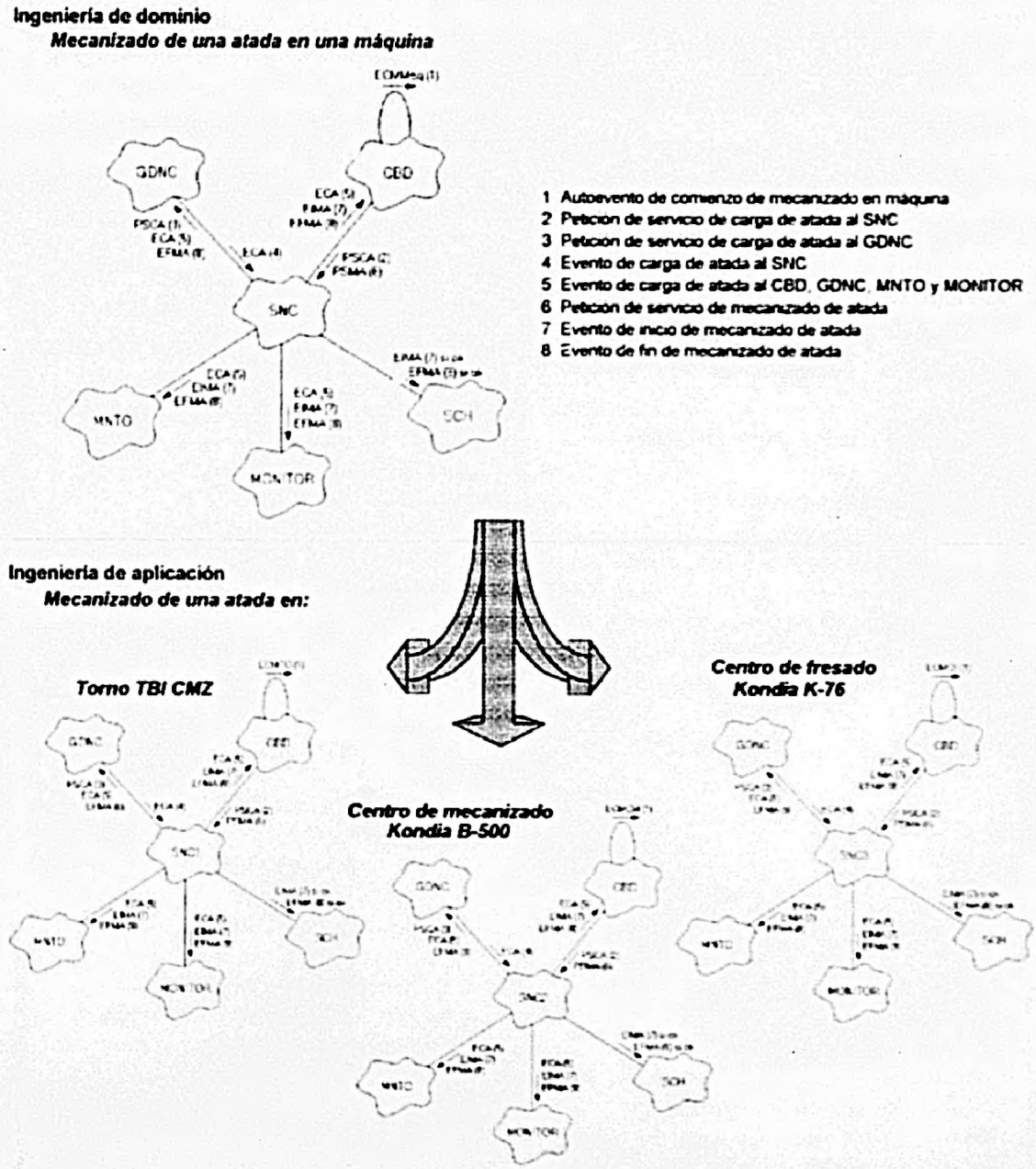


Figura 4.10: Casos particulares de la tarea de mecanizado de una atada.

La existencia de un lector de códigos de herramienta en el centro de mecanizado Kondia B500 provoca un ligero cambio en el MFD de la tarea de carga de herramientas desde el almacén intermedio para dicha máquina. La acción de introducir el código de herramienta que realiza el operario del dominio se convierte en una acción opcional que únicamente se realizará cuando falle la introducción automática del código.

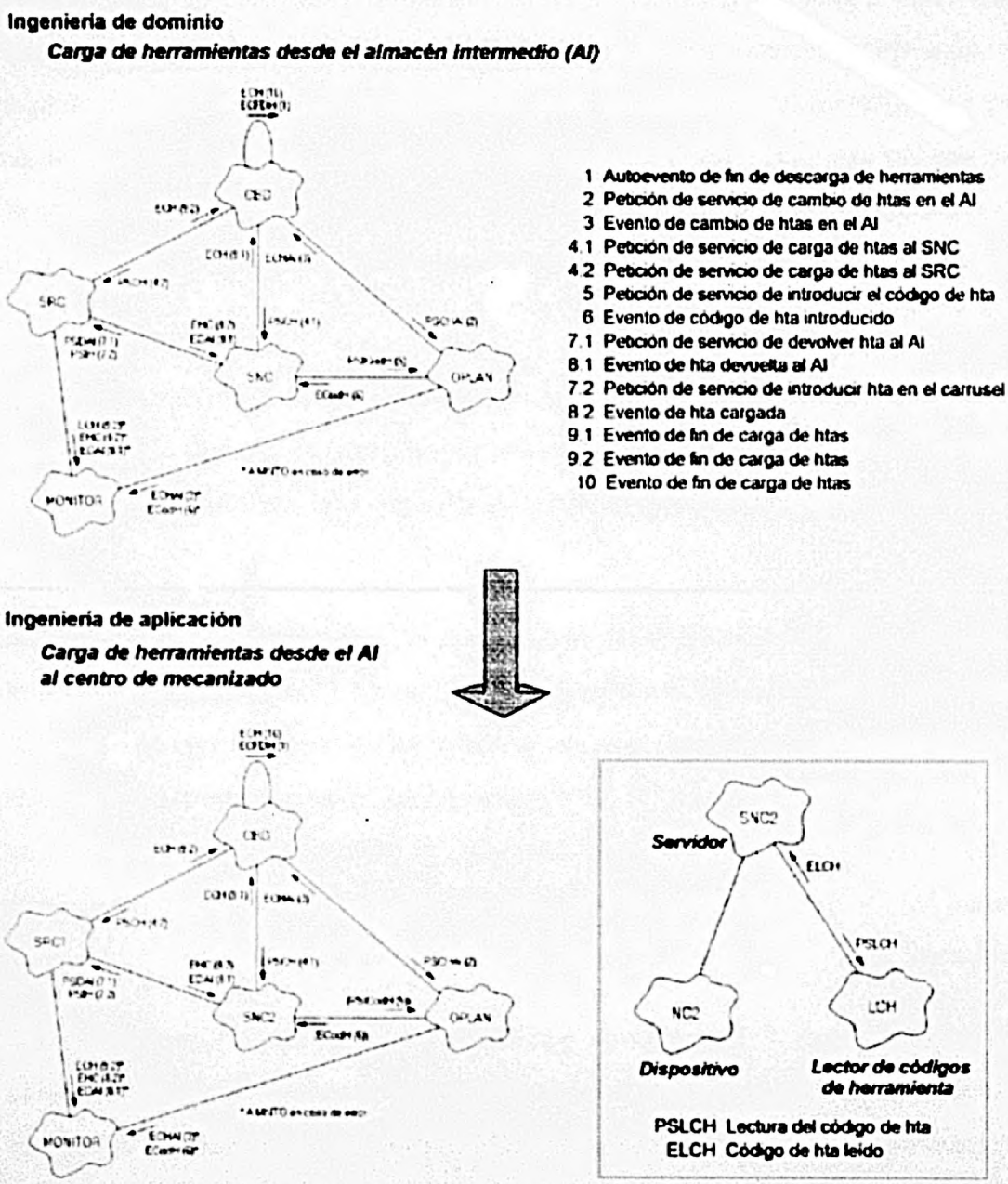


Figura 4.11: Tarea de carga de herramientas desde el almacén intermedio para el DCM1.

Sin embargo, las repercusiones son mayores en la etapa de implementación, puesto que es necesario un driver que gobierne el lector de códigos. Por lo tanto, el servidor SNC2 debe controlar dos drivers (el driver del control numérico y el driver del lector de códigos).

También se producen modificaciones en el MFD de la tarea de operaciones de mantenimiento, debido a la existencia de un sistema de adquisición de datos (DAS – Data Acquisition System) que monitoriza las fuerzas de corte y las vibraciones del torno. La adaptación ha consistido en la *inclusión de este nuevo sistema* en el MFD, y de los dos mensajes que emite al módulo Mantenimiento on-line con la información de dichos valores. Los dos nuevos mensajes están perfectamente descritos de acuerdo al modelo de eventos, y acatan el estándar de mensajería entre procesos impuesto por el dominio.

Pero sin duda alguna, *los MFDs que requieren mayor esfuerzo de adaptación son los correspondientes a las tareas de ejecución de un plan de trabajo y reserva de herramientas*, que pueden requerir la inclusión de algún servidor con su secuencia de mensajes.

No obstante, la adaptación del MFD de ejecución de un plan de trabajo ha supuesto simplemente la sustitución de identificadores (SNC por SNC2). Al finalizar un plan, el CBD solicita al gestor de herramientas que actualice los valores de la vida remanente de las mismas en la base de datos. En el proyecto piloto, sólo el centro de mecanizado Kondia B500 tiene capacidad para controlar el tiempo de utilización de las herramientas, y por lo tanto, únicamente su servidor puede proporcionar esta información.

En el caso de reserva de herramientas, además de la sustitución de identificadores, se han añadido dos servidores de máquinas de control numérico que únicamente requieren los mensajes de carga la lista de correctores. Como estas máquinas realizan la carga de herramientas de forma manual, es el operario el que debe recibir la lista con las herramientas que deben ser cargadas, y no los servidores de estos dispositivos.

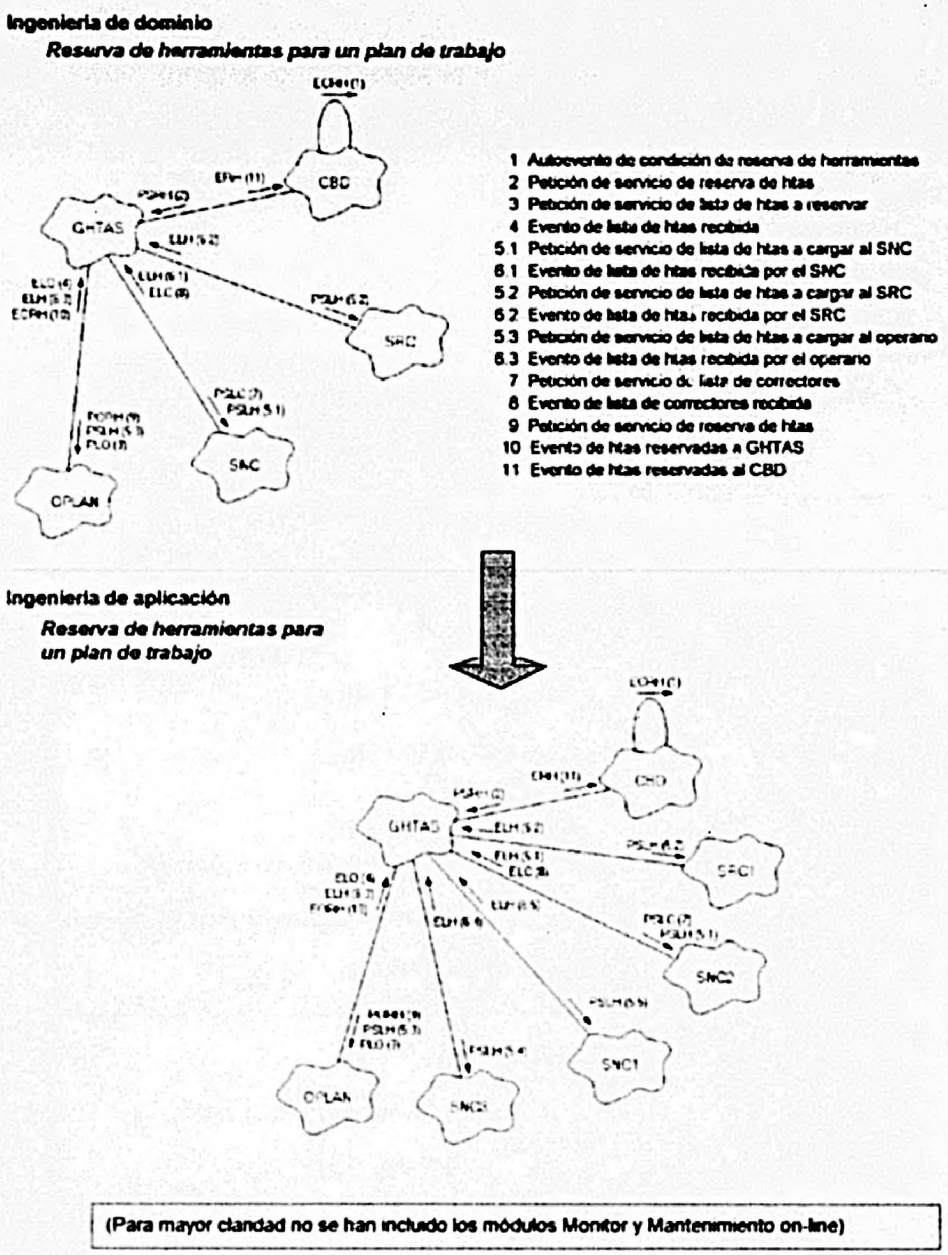


Figura 4.12: Tarea de reserva de herramientas para un plan de trabajo.

4.3.2.2.1 Descripción de los mensajes.

Las *relaciones entre módulos quedan perfectamente identificadas en los MFDs* por medio de los mensajes que intercambian y la secuencia de dicho intercambio. Además, todos y cada uno de *los mensajes disponen de una descripción detallada,* que se ajusta al modelo de petición de servicio o al modelo de evento, según el tipo de mensaje.

El contenido de estos modelos (establecido en la ingeniería de dominio) permite *definir las interfaces de los mensajes con independencia del lenguaje de programación* que se use en su codificación.

Salvo los dos mensajes nuevos del sistema DAS, las descripciones de las restantes peticiones de servicio y eventos se han obtenido del dominio con la simple sustitución de algunos identificadores.

Cabecera del evento		
Módulo:	SNC1	
Evento:	ESNC1_FinMecanizadoAtada()	
Id_evento:	ESNC1_FinMecanizadoAtada	
Versión:	1	
Condición:	Fin del mecanizado de una atada en el torno	
Cuerpo del evento		
{INT32 ESNC1_FinMecanizadoAtada (TListaProg [in] ListaProg, TError [in] Error)}		
Lista de receptores del evento		
GDNC	DAE	Fin mecanizado
CBD	TCP/IP (Brigde G2)	Fin mecanizado
MONITOR	TCP/IP	Fin mecanizado
SCH	TCP/IP (Brigde G2)	Mecanizado OK
MNTO	TCP/IP	En caso de error

Tabla 4.7: Descripción de un evento.

Cabecera del servicio	
Módulo:	SNC1
Servicio:	SNC1_MecanizaAtada()
Id_servicio:	SNC1_MecanizaAtada
Interface:	DAE
Versión:	1
Clientes:	CBD
Descripción:	Petición del mecanizado de una atada en el torno
Cuerpo del servicio	
Asincrono [ON, OFF, EVENT= ESNC1_InicioMecanizadoAtada, ESNC1_FinMecanizadoAtada] {INT32 SNC1_MecanizaAtada (TListaProg [in] ListaProg)}	

Tabla 4.8: Descripción de una petición de servicio.

4.3.2.3 Especificaciones de mensajería de los módulos.

A partir de las especificaciones de mensajería del dominio para los módulos genéricos y de los MFDs particulares de esta aplicación, se construyen las especificaciones de mensajería de los módulos identificados en la arquitectura del proyecto piloto: *se eliminan aquellas peticiones de servicio y eventos que no se empleen, y se añaden los mensajes nuevos.*

Las especificaciones de mensajería de los distintos módulos permiten identificar el *conjunto de mensajes que deben procesar, que se traducen en funciones* en la etapa de implementación.

A continuación se define la especificación de mensajería de cada módulo:

- **Especificación de mensajería del módulo Monitor.** El módulo Monitor recibe *todos los eventos que se generan en el sistema,* aunque no todos ellos implican la actualización de la representación gráfica de la planta.

- **Especificación de mensajería del módulo Mantenimiento on-line.** El módulo Mantenimiento on-line recibe *todos los eventos con error que se generen en el sistema.* También se han incluido los eventos emitidos por el sistema DAS del torno: EDAS_ErrorFuerzasCorte y EDAS_ErrorVibraciones.

- **Especificación de mensajería del módulo Scheduler dinámico.** La especificación del dominio para el Scheduler ha sido modificada con la *copia de dos eventos* de finalización del mecanizado de una atada (tiene un evento de este tipo por cada máquina CN), y la *adecuación de los identificadores de los servidores* que emiten el evento.

- **Especificación de mensajería del módulo Control básico Dispatcher.** Las modificaciones en la especificación del CBD vienen dadas por el *conjunto de peticiones de servicio y eventos que se transmiten hacia desde los servidores* de dispositivos.

- **Especificación de mensajería de los módulos gestores de recursos.** La especificación del **gestor de herramientas** ha requerido la *sustitución de identificadores* para los mensajes relacionados con la actualización y reserva de herramientas para posterior carga/descarga automática en el DCM1, y la *replica con modificación de identificadores* para la carga de la lista de correctores en las tres máquinas de mecanizado. También *se han eliminado* todos los mensajes asociados con el paletizado, carga/descarga desde palet y despaletizado de herramientas.

  En el caso del **gestor de programas CN,** la especificación del dominio ha sido modificada con la *copia y modificación de identificadores* para los eventos de finalización de la carga inicial de programas, carga de una atada, y final del mecanizado de una atada, procedentes de las tres máquinas de CN.

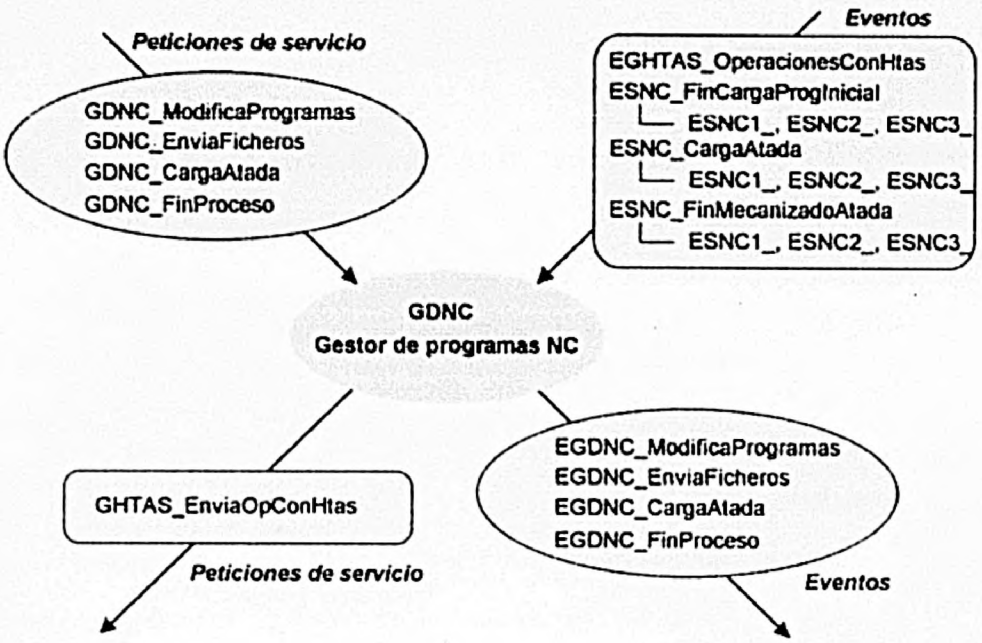


Figura 4.13: Mensajes del gestor de programas CN.

- **Especificación de mensajería de los módulos Servidores de dispositivos.** Las modificaciones que se han realizado en las especificaciones del dominio para los distintos servidores de dispositivos son:
  - **SNC1 (servidor del torno).** Sustitución del identificador SNC por SNC1, y eliminación de todos aquellos mensajes relacionados con la carga/descarga automática de herramientas y su actualización. También se ha ampliado la lista de clientes de los eventos inicio/fin de atada con la inclusión del sistema DAS.
  - **SNC2 (servidor del centro de mecanizado).** Sustitución del identificador SNC por SNC2, y supresión de mensajes para carga/descarga manual de herramientas.
  - **SNC3 (servidor del centro de fresado).** Sustitución del identificador SNC por SNC3, y eliminación de los mensajes asociados con la carga/descarga automática de herramientas y su actualización.
  - **SRC1 (servidor del Fanuc S-700).** Sustitución del identificador SRC por SRC1, y eliminación de los mensajes propios de las tareas de paletizado,

despaletizado, carga de programas y carga/descarga de herramientas desde palet.

- **SRC2 (servidor del Fanuc S-10).** Sustitución del identificador SRC por SRC2, y eliminación de los mensajes correspondientes a la carga/descarga de herramientas, la carga de programas y el despaletizado.
- **SRC3 (servidor del Asea IRB6).** Sustitución del identificador SRC por SRC3, y eliminación de los mensajes para realizar las tareas de carga/descarga de herramientas, carga de programas, paletizado y despaletizado.
- **SPLC (servidor del sistema de transporte).** Sin modificaciones.
- **OPLAN (servidor del operario).** Eliminación de los mensajes relacionados con la carga/descarga de piezas.

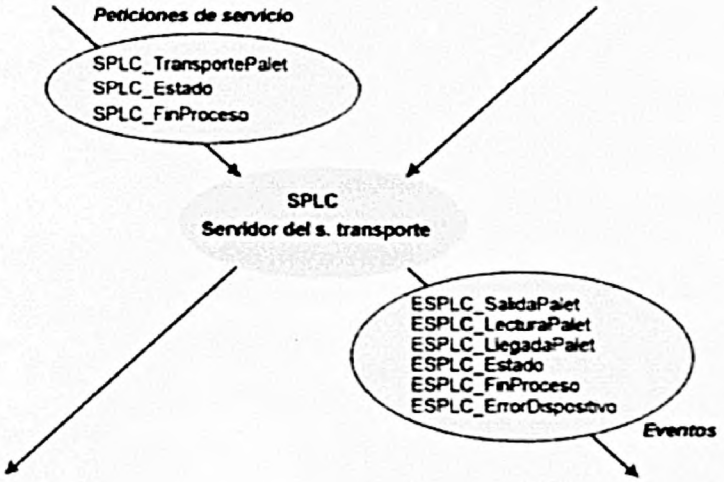


Figura 4.14: Mensajes del servidor del sistema de transporte.

4.3.2.3.1 Información adicional para la etapa de implementación.

Además de las especificaciones de los distintos módulos, en la etapa de implementación se necesitan una serie de datos sobre **el equipo informático** (modelo, sistema operativo y versión), **la plataforma de integración** (nombre y versión), **el entorno de desarrollo** (entorno, lenguaje de programación y versión), **la red** (tipo, dirección IP, nombre del

nodo, velocidad, protocolo y posibilidades de conexión), y **las conexiones con los dispositivos de planta** (hardware y software), que se recogen en *una tabla por módulo.*

Nombre del módulo	CBD	
Descripción	Control básico Dispatcher	
Equipo informático		
Modelo	32h RS/6000 de IBM	
Sistema operativo	AIX	
Versión	3.2.5	
Plataforma de integración	☐ Si	☑ No
Nombre		
Versión		
Entorno de desarrollo		
Entorno	G2	
Lenguaje programación	GRAFCHART	
Versión		
Red		
Tipo	Token-Ring	
Dirección IP	158.227.66.110	
Nombre del nodo	Ibm1, g2, grasp, rs32h	
Velocidad	16 Mbps	
Protocolo	TCP/IP	
Posibilidades conexión	GSI	
Conexión con la planta	☐ Si	☑ No
Hardware		
Software		

Tabla 4.9: Información del módulo CBD para la etapa de implementación.

4.3.2.4 Verificación de la arquitectura.

Una vez realizada la *verificación de la arquitectura contra la especificación de la aplicación particular*, queda concluida la fase de diseño y se inicia la de implementación.

No es necesario verificar las arquitecturas de los módulos, porque esta tarea se ha realizado en el proceso de ingeniería de dominio.

4.3.3 Implementación de la aplicación.

Las tareas que se realizan en esta etapa son las siguientes: **construir los módulos** identificados en la arquitectura a partir de los componentes generados en el proceso de ingeniería de dominio, e **integrarlos** para obtener el sistema final.

4.3.3.1 Construcción de los módulos del nuevo sistema.

Los procesos de construcción de los módulos necesarios en el proyecto piloto están descritos en la sección "Implementación del dominio" del presente capítulo.

A continuación se presentan las *acciones realizadas para obtener los módulos particulares atendiendo a los siguientes criterios:*

- *Sustitución de los identificadores del dominio.*
- *Eliminación de funciones que implementan mensajes no contemplados en su especificación.*
- *Replica de aquellas funciones que implementan el mismo mensaje pero con distintos orígenes o destinos.*
- *Adaptación de las tablas de relación y datos de estado.*
- *Desarrollo de nuevos componentes.*

4.3.3.1.1 Construcción del módulo Monitor.

Acciones realizadas:

- **Sustitución de identificadores.** Se han sustituido los identificadores SNC por SNC1, SNC2 y SNC3, y SRC por SRC1, SRC2 y SRC3 en las funciones replicadas.
- **Eliminación de funciones.** Se han eliminado todas aquellas funciones que tratan peticiones de servicio no atendidas por los servidores de dispositivos y eventos no emitidos por los mismos.
- **Replica de funciones.** Se han replicado las funciones de tratamiento de las peticiones de servicio que pueden recibir los servidores, y de los eventos procedentes de los servidores de máquinas de control numérico y de robots.
- **Adaptación de tablas de relación.** No se han hecho cambios en los datos de estado, y se han empleado todas las tablas de relación y ficheros sin modificación del formato.
- **Nuevos desarrollos.** Desarrollo del componente de representación gráfica con ayuda de rutinas gráficas para el trazado de los distintos elementos.

4.3.3.1.2 Construcción del módulo Mantenimiento on-line.

Acciones realizadas:

- **Sustitución de identificadores.** Se han sustituido los identificadores SNC por SNC1, SNC2 y SNC3, y SRC por SRC1, SRC2 y SRC3 en las funciones replicadas.
- **Eliminación de funciones.** Se han eliminado todas aquellas funciones que tratan eventos no emitidos por los servidores de dispositivos.
- **Replica de funciones.** Se han replicado las funciones de tratamiento de los eventos procedentes de los servidores de máquinas CN y de robots.

- **Adaptación de tablas de relación.** No se han hecho cambios en los datos de estado, y se han empleado todas las tablas de relación y ficheros sin modificación del formato.

- **Nuevos desarrollos.** Desarrollo del software específico de monitorización continua. Este sistema dispone de una tarjeta de adquisición de datos analógico/digital (AT-MIO-16 de National Instruments) que recoge las señales procedentes del monitor de herramientas TS200 Tool Monitor de Montronix, y del acelerómetro colocado en el cabezal del torno.

  El tratamiento de las señales recogidas así como la generación y envío del posible mensaje de anomalía, se realiza con el software de adquisición y tratamiento de señales DADISP de Novatronik. Este software se ejecuta en un *PC bajo sistema operativo MS-DOS* y se comunica con los restantes módulos a través del *protocolo TCP/IP*.

  Además, se han añadido dos funciones en el módulo para implementar los eventos procedentes del sistema DAS.

Observaciones. El componente diagnóstico se ha reutilizado de trabajos previos. Pero este dato no es significativo, porque se trata de un desarrollo específico con gran dependencia de la planta de fabricación.

4.3.3.1.3 Construcción del módulo Scheduler dinámico.

Acciones realizadas:

- **Sustitución de identificadores.** Se ha reemplazado el identificador SNC por SNC1, SNC2 y SNC3 en los procedimientos replicados.

- **Eliminación de funciones.** No se ha realizado.

- **Replica de funciones.** Se han triplicado las funciones que procesan el evento de fin de mecanizado de una atada.

- **Adaptación de tablas de relación.** No se han hecho cambios en los datos de estado, y se han usado las tablas de relación y ficheros sin modificar su formato.
- **Nuevos desarrollos.** No se han realizado.

Observaciones. La base de conocimiento se han modificado para incluir todos los mensajes que intercambia el Scheduler en las clases de objetos interface.

4.3.3.1.4 Construcción del módulo Control básico Dispatcher.

Acciones realizadas:

- **Sustitución de identificadores.** Se han sustituido los identificadores SNC por SNC1, SNC2 y SNC3, y SRC por SRC1, SRC2 y SRC3. Esta tarea se ha realizado diagrama por diagrama.
- **Eliminación de funciones.** La eliminación de procedimientos G2 está asociada con la supresión de etapas en los distintos diagramas:
  - *Diagrama principal.* Se han eliminado las etapas de carga inicial de programas en los robots contenidas en la macro de carga inicial de herramientas y programas.
  - *Diagrama de ruta.* Se han eliminado las etapas de carga de programas en robot de la macro de carga herramientas y programas, las etapas de carga manual en la macro de carga, las etapas de descarga manual en la macro de descarga, y las etapas de despaletizado automático de la macro de despaletizado.
  - *Diagrama de workstation de paletizado.* No se han eliminado etapas.
  - *Diagrama de workstation de torno.* Se han eliminado las etapas de carga/descarga manual de piezas y carga/descarga automática de herramientas.
  - *Diagrama de workstation de centro de mecanizado 1.* Se han eliminado las etapas de carga/descarga manual de piezas y de herramientas.

- *Diagrama de workstation de centro de mecanizado 2.* Se han eliminado las etapas de carga/descarga manual de piezas y carga/descarga automática de herramientas.
- *Diagrama de workstation de transporte.* No se han eliminado etapas.
- *Diagrama de workstation de despaletizado.* Se han eliminado las etapas de despaletizado automático de herramientas.
- *Diagrama de gestor de herramientas.* Se han eliminado las etapas de paletizado, carga/descarga desde palet y despaletizado de herramientas.
- *Diagrama de gestor de programas CN.* No se han eliminado etapas.
- *Diagrama de servidor del torno.* Se han eliminado las etapas de actualización y carga/descarga automática de herramientas.
- *Diagrama de servidor de centro de mecanizado.* Se han eliminado las etapas de carga/descarga manual de herramientas.
- *Diagrama de servidor de centro de fresado.* Se han eliminado las etapas de actualización y carga/descarga automática de herramientas.
- *Diagrama de servidor de robot Famuc S-700.* Se han eliminado las etapas de paletizado, despaletizado, carga de programas y carga/descarga de herramientas desde palet.
- *Diagrama de servidor de robot Famuc S-10.* Se han eliminado las etapas de carga de programas, carga/descarga de herramientas y despaletizado.
- *Diagrama de servidor de robot Asea IRB6.* Se han eliminado las etapas de carga de programas, carga/descarga de herramientas, paletizado, y despaletizado.
- *Diagrama de servidor de sistema de transporte.* No se han eliminado etapas.
- *Diagrama de servidor de operario.* Se han eliminado las etapas de carga/descarga manual de piezas.

- **Replica de funciones.** Se han duplicado los diagramas de workstation de centro de mecanizado, y triplicado los diagramas de servidores de control numérico y de robot, y con ellos los procedimientos G2 asociados. En cuanto a

procedimientos, se han triplicado las etapas de modificación de los programas CN y de carga de correctores en los diagramas de los gestores.

- **Adaptación de tablas de relación.** No se han hecho cambios en los datos de estado, y se han usado las tablas de relación y ficheros sin modificar su formato.
- **Nuevos desarrollos.** No se han realizado.

Observaciones. La base de conocimiento se ha modificado para incluir todos los mensajes que intercambia el CBD en las clases de objetos interface.

4.3.3.1.5 Construcción del módulo gestor de herramientas (GHTAS).

Acciones realizadas:

- **Sustitución de identificadores.** Se ha reemplazado SRC por SRC1, SNC por SNC2 en las funciones de actualización y reserva de herramientas para carga/descarga automática, y SNC por SNC1, SNC2 y SNC3 en las funciones replicadas de carga de la lista de correctores.
- **Eliminación de funciones.** Se han eliminado las funciones para las tareas de paletizado, carga/descarga desde palet y despaletizado de herramientas (15 funciones borradas de un total de 39).
- **Replica de funciones.** Se han realizado dos copias de la petición de servicio y el evento correspondientes a la carga de la lista de correctores, de modo que todas las máquinas de mecanizado reciben la petición y el gestor de herramientas procesa los eventos enviados por estos dispositivos.
- **Adaptación de tablas de relación.** No se han hecho cambios en los datos de estado, y se han empleado todas las tablas de relación.
- **Nuevos desarrollos.** No se han realizado.

Observaciones. Los datos de las herramientas existentes en el sistema se han introducido en la base datos.

4.3.3.1.6 Construcción del módulo gestor de programas de control numérico (GDNC).

Acciones realizadas:

- **Sustitución de identificadores.** Se ha reemplazado SNC por SNC1, SNC2 y SNC3 en las funciones replicadas.
- **Eliminación de funciones.** No se ha realizado.
- **Replica de funciones.** Se han triplicado las funciones que procesan los eventos de finalización de la carga inicial de programas, de carga de una atada y de finalización del mecanizado de una atada, de modo que el gestor puede realizar el tratamiento de los correspondientes eventos procedentes de las tres máquinas de mecanizado.
- **Adaptación de tablas de relación.** Se ha construido el árbol de directorios necesario para esta aplicación y se han insertado los programas tipo de cada una de las máquinas.
- **Nuevos desarrollos.** No se han realizado.

Observaciones. Se ha introducido la información referente al presente proyecto de ingeniería de aplicación en la base de datos.

4.3.3.1.7 Construcción del módulo servidor del torno (SNC1).

Acciones realizadas:

- **Sustitución de identificadores.** Se ha reemplazado SNC por SNC1.
- **Eliminación de funciones.** Se han borrado las funciones de actualización y carga/descarga automática de herramientas (12 funciones eliminadas de un total de 38).
- **Replica de funciones.** No se ha realizado.

- **Adaptación de tablas de relación.** Los datos de estado no se han modificado. No ha sido necesaria la tabla de relación que asocia las posiciones de la torreta con los programas CN que las mueven a las posiciones de carga/descarga debido a que esta tarea no se realiza por cada plaquita, sino en conjunto.

- **Nuevos desarrollos.** No se han realizado, porque el driver para el control numérico FAGOR 8050F se ha reutilizado de trabajos previos.

Observaciones. Las funciones que emiten los eventos de inicio y finalización de mecanizado de una atada se han modificado para incluir el envío de estos eventos al sistema DAS.

4.3.3.1.8 Construcción del módulo servidor del centro de mecanizado (SNC2).

Acciones realizadas:

- **Sustitución de identificadores.** Se ha reemplazado SNC por SNC2 y SRC por SRC1.

- **Eliminación de funciones.** Se han borrado las funciones asociadas con las operaciones de carga/descarga manual de herramientas (6 funciones eliminadas de un total de 38).

- **Replica de funciones.** No se ha realizado.

- **Adaptación de tablas de relación.** No se han hecho cambios en los datos de estado, y se han empleado todas las tablas de relación.

- **Nuevos desarrollos.** No se han realizado, porque los drivers para el control numérico Fidia Copymill y para el lector de códigos de herramientas se han reutilizado de trabajos previos.

Observaciones. Las funciones correspondientes a la inserción del código de herramienta han sido modificadas para incluir el control del driver para el lector de códigos, el cual permite realizar dicha tarea de forma automática.

4.3.3.1.9 Construcción del módulo servidor del centro de fresado (SNC3).

Acciones realizadas:

- **Sustitución de identificadores.** Se ha reemplazado SNC por SNC3.
- **Eliminación de funciones.** Se han borrado las funciones de actualización y carga/descarga automática de herramientas (12 funciones eliminadas de 38).
- **Replica de funciones.** No se ha realizado.
- **Adaptación de tablas de relación.** No se han hecho cambios en los datos de estado, y se han empleado todas las tablas de relación.
- **Nuevos desarrollos.** No se han realizado, porque el driver para el control numérico FAGOR 8010M se ha reutilizado de trabajos previos.

4.3.3.1.10 Construcción del módulo servidor del robot Fanuc S-700 (SRC1).

Acciones realizadas:

- **Sustitución de identificadores.** Se ha reemplazado SRC por SRC1.
- **Eliminación de funciones.** Se han borrado las funciones de paletizado, despaletizado, carga de programas y carga/descarga de herramientas desde palet (11 funciones eliminadas de un total de 21).
- **Replica de funciones.** No se ha realizado.
- **Adaptación de tablas de relación.** Todos los datos de estado y tablas de relación se han utilizado.
- **Nuevos desarrollos.** No se han realizado, porque el driver se ha reutilizado de trabajos previos.

Observaciones. Para cualquier servidor de robot, rellenar la tabla de relación entre las acciones y la secuencia de programas a ejecutar es un punto crítico, porque requiere el suministro de estas secuencias para su construcción. Esta tabla determina qué operaciones puede realizar el robot.

4.3.3.1.11 Construcción del módulo servidor del robot Fanuc S-10 (SRC2).

Acciones realizadas:

- **Sustitución de identificadores.** Se ha reemplazado SRC por SRC2.
- **Eliminación de funciones.** Se han borrado las funciones de las tareas de carga de programas, carga/descarga de herramientas y despaletizado (9 funciones eliminadas de un total de 21).
- **Replica de funciones.** No se ha realizado.
- **Adaptación de tablas de relación.** Los datos de estado se han utilizado, y de las tablas de relación no se ha empleado la que asocia los códigos de herramientas con las garras que las manipulan, puesto que este robot no realiza carga/descarga de herramientas.
- **Nuevos desarrollos.** No se han realizado, porque el driver se ha reutilizado de trabajos previos.

4.3.3.1.12 Construcción del módulo servidor del robot Asea IRB6 (SRC3).

Acciones realizadas:

- **Sustitución de identificadores.** Se ha reemplazado SRC por SRC3.
- **Eliminación de funciones.** Se han borrado las funciones correspondientes a la carga de programas, carga/descarga de herramientas, paletizado, despaletizado y cambio de garras (12 funciones eliminadas de un total de 21).
- **Replica de funciones.** No se ha realizado.
- **Adaptación de tablas de relación.** El dato de estado "situación de las garras" no se ha usado porque este robot no tiene la posibilidad de cambio de garras. Por la misma razón, tampoco se ha utilizado la tabla que relaciona los códigos de pieza y las garras que las manipulan. Como en el caso del SRC2, no se ha

empleado la tabla que asocia los códigos de herramientas con las garras que las manipulan, puesto que este robot no realiza carga/descarga de herramientas.

- **Nuevos desarrollos.** No se han realizado, porque el driver se ha reutilizado de trabajos previos.

4.3.3.1.13 Construcción del módulo servidor del sistema de transporte (SPLC).

Acciones realizadas:

- **Sustitución de identificadores.** No se ha realizado.
- **Eliminación de funciones.** No se ha realizado.
- **Replica de funciones.** No se ha realizado.
- **Adaptación de tablas de relación.** Los datos de estado y la tabla de relación son los mismos.
- **Nuevos desarrollos.** No se han realizado, porque el driver para el PLC Simatic S5-115U se ha reutilizado de trabajos previos.

Observaciones. El programa del PLC es el determinante de las operaciones que puede realizar el sistema de transporte.

4.3.3.1.14 Construcción del módulo servidor de operario (OPLAN).

Acciones realizadas:

- **Sustitución de identificadores.** No se ha realizado.
- **Eliminación de funciones.** Se han eliminado las funciones correspondientes a la carga/descarga manual de piezas (4 funciones eliminadas de un total de 34).
- **Replica de funciones.** No se ha realizado.
- **Adaptación de tablas de relación.** No se ha realizado.
- **Nuevos desarrollos.** No se han realizado.

4.3.3.1.15 Construcción del bus de mensajes.

Acciones realizadas:

- **Definición de los identificadores** de las peticiones de servicio y de los eventos, de modo que dichos identificadores sean *únicos* en el sistema.
- **Construcción de los bridges necesarios** con el generador.

4.3.3.2 Construcción del sistema final.

El sistema final se obtiene como *resultado del proceso de integración* de todos los módulos implementados y probados. Las pruebas se han realizado siguiendo una *estrategia incremental ascendente en las distintas workstations.*

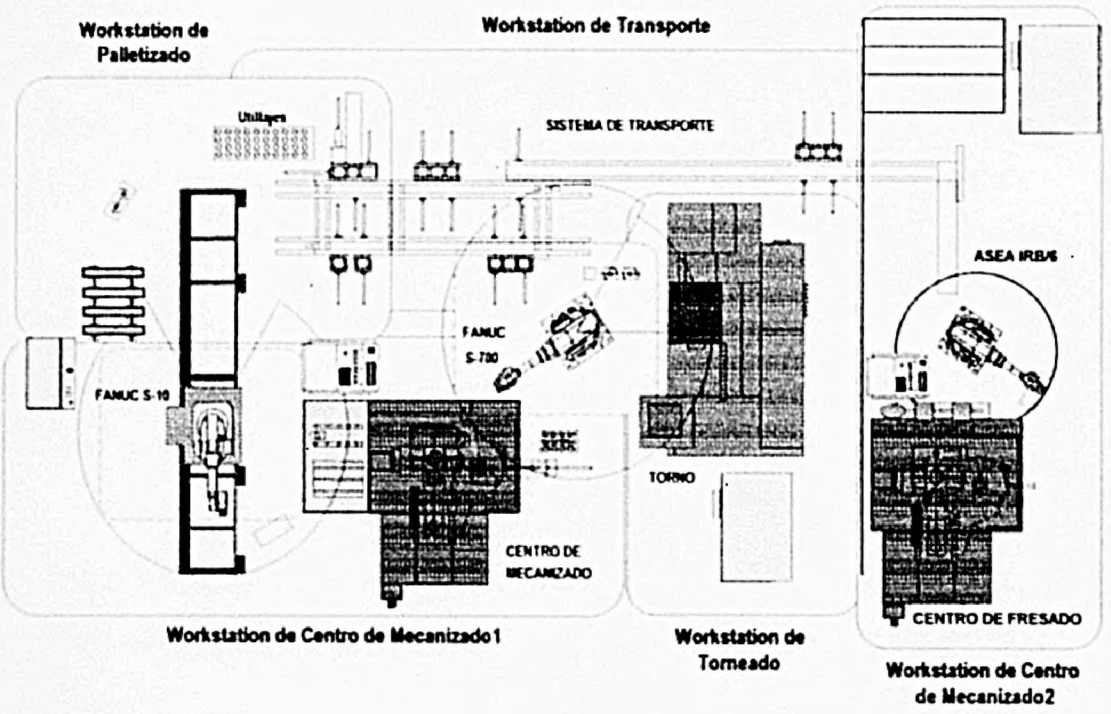


Figura 4.15: División en workstations[39] de la planta de fabricación.

[39] No está delimitada la workstation de despaletizado porque se corresponde con el tobogán de salida de piezas del sistema. Cuando llega el palet, el operario retira la pieza y el palet.

4.3.3.2.1 Integración por áreas funcionales o workstations.

El orden del proceso de integración en las distintas workstations se describe seguidamente:

- **Workstation de paletizado.** La integración se ha realizado en el siguiente orden: SRC2 (servidor del robot Fanuc S-10) y SPLC (servidor del sistema de transporte), y posteriormente, los módulos: OPLAN (servidor de operario), CBD, Monitor, Mantenimiento on-line y Scheduler dinámico de forma incremental.

- **Workstation de torno.** SNC1 (servidor del torno) y SRC1 (servidor del robot Fanuc S-700) se han integrado en primer lugar. Luego se han añadido: SPLC, OPLAN, GHTAS (gestor de herramientas), GDNC (gestor de programas CN), CBD, Monitor, Mantenimiento on-line y Scheduler dinámico de forma incremental.

- **Workstation de centro de mecanizado 1.** La integración comienza con SNC2 (servidor del centro de mecanizado) y SRC2, y se van añadiendo de forma incremental los siguientes módulos: SPLC, SRC1, OPLAN, GHTAS, GDNC, CBD, Monitor, Mantenimiento on-line y Scheduler dinámico.

- **Workstation de centro de mecanizado 2.** El orden de la integración ha sido el siguiente: SNC3 (servidor del centro de fresado) con SRC3 (servidor del robot Asea IRB6), SPLC, OPLAN, GHTAS, GDNC, CBD, Monitor, Mantenimiento on-line y Scheduler dinámico.

- **Workstation de transporte.** SPLC se ha integrado con el CBD, y posteriormente se han añadido los módulos Monitor, Mantenimiento on-line y Scheduler dinámico de forma incremental.

- **Workstation de despaletizado.** El servidor de operario (OPLAN) se ha integrado con el CBD, y se han sumado Monitor, Mantenimiento on-line y Scheduler dinámico de forma incremental.

4.3.3.3 Verificación y validación del sistema.

Las pruebas finales han consistido en la *ejecución de un plan de trabajo*, que ha permitido probar las distintas tareas identificadas en la operativa del sistema y los requisitos funcionales solicitados.

4.4 Conclusiones.

El grado de reutilización se ha evaluado para cada una de las etapas del proceso de desarrollo y para cada módulo. Los **resultados generales obtenidos** son los siguientes:

- **Las especificaciones, las arquitecturas, los casos de prueba y los componentes de prueba presentan un elevado grado de reutilización (90-100%).** Las modificaciones que se han realizado en estos elementos son mínimas, y siempre debidas a la incorporación de algún dispositivo poco habitual en un FMS (lector de códigos de herramientas y sistema de monitorización continua).

- **Para los componentes implementados, se ha constatado que la valoración del grado de reutilización varía en función del tipo de módulo:**

  - *Monitor.* Presenta un *bajo nivel de reutilización*, debido a que la representación gráfica depende del layout de la planta. Además, la adaptación de los componentes gestor de peticiones de usuario y gestor de servicios y eventos es una tarea laboriosa (aunque no complicada), porque deben procesar todas los mensajes que puede intercambiar con los servidores de los dispositivos.
  - *Mantenimiento on-line.* También presenta un *bajo nivel de reutilización*, porque el componente diagnóstico necesita los FMEAs y FTAs de la planta de fabricación, que varían según los dispositivos existentes y la combinación de los mismos.

    Por otro lado, los desarrollos específicos de los componentes de monitorización continua van a añadir un conjunto de mensajes (más o menos numeroso) que deben ser procesados.

    A esto hay que añadir que la adaptación del componente gestor de servicios y eventos es laboriosa, debido a que debe procesar todos los eventos con error que ocurran en el sistema.
  - *Scheduler dinámico.* Presenta un *alto nivel de reutilización para la presente aplicación.* Pero la adecuación de las estrategias para tratar determinadas

situaciones (alta de un pedido, cancelación, cambio de prioridad, etc.) puede requerir un esfuerzo significativo en otras aplicaciones. Por lo tanto, se considera con un *nivel medio de reutilización.*

- *Control básico Dispatcher.* Presenta un *nivel de reutilización medio*, porque la adaptación de los diagramas es una tarea laboriosa.
- *Gestores de recursos.* Presentan un *alto nivel de reutilización*, siempre y cuando la base de datos sea Access. De lo contrario, sería necesario el desarrollo del driver para la base de datos en cuestión.
- *Servidores de dispositivos.* Presentan un *alto nivel de reutilización* cuando los driver necesarios están disponibles. Los procesos de adaptación para estos módulos son muy sencillos, y el componente espía es reutilizable al 100%.

También **presenta un alto nivel de reutilización el generador de bridges,** que permite construir automáticamente el software de comunicaciones para resolver los problemas de conectividad de los módulos CBD y Scheduler dinámico.

Los **resultados obtenidos han sido satisfactorios,** pese a los niveles de reutilización obtenidos en los componentes implementados para algunos módulos de control (Monitor y Mantenimiento on-line). Además, **los niveles de reutilización obtenidos en las etapas de análisis y diseño ya constatan el éxito de la metodología propuesta.**

Conclusiones

El trabajo presentado propone **una nueva metodología para la construcción del software de control para FMS,** entroncada *en el enfoque de la Megaprogramación.* Desde el inicio del propio proceso de desarrollo, esta metodología *incorpora la reutilización de forma sistemática,* y se fundamenta en la *distinción entre ingeniería de dominio e ingeniería de aplicación,* al concebir un software reutilizable.

El éxito de esta metodología reside en la medida que la ingeniería de aplicación interactúe fuertemente con la ingeniería de dominio. Existe una fuerte interacción entre ambas cuando la mayoría de los componentes empleados para desarrollar una aplicación particular residen en el repositorio, y es mínimo el esfuerzo requerido para una posible adaptación.

El grado de reutilización del software de control desarrollado utilizando esta metodología, **se ha evaluado en un proyecto piloto desarrollado en el Departamento de Ingeniería Mecánica de la Escuela Técnica Superior de Ingenieros de la UPV/EHU** (Universidad del País Vasco / Euskal Herriko Unibertsitatea), en el bienio 1.996-97. La evaluación se ha realizado *para cada una de las etapas del proceso de construcción* de una aplicación particular, y también *para los distintos módulos implementados.*

Los resultados son satisfactorios, sobre todo **para las etapas de análisis y diseño que presentan una fuerte interacción entre la ingeniería de dominio y la ingeniería de aplicación.**

Para la etapa de implementación, los componentes empleados en la construcción de los módulos particulares residen en el repositorio, pero **el esfuerzo de adaptación necesario depende del tipo de módulo:** *mínimo para los gestores de recursos y servidores de dispositivos, medio para el Scheduler dinámico y el Control básico Dispatcher, y elevado para el Monitor y Mantenimiento on-line.* No obstante, estos esfuerzos medios y elevados de adaptación, son inferiores al esfuerzo de crear el módulo partiendo de cero.

La aplicación práctica del trabajo presentado **reportará beneficios a las pequeñas y medianas empresas manufactureras,** ya que supone un gran paso en la resolución de dos graves problemas que acusa la construcción del software de control de FMS: el coste de desarrollo y el tiempo de puesta en marcha.

Esta metodología brinda la posibilidad de utilizar los mismos componentes en distintas aplicaciones o adaptar algunos a las necesidades específicas con un mínimo esfuerzo, **sin tener que desarrollar íntegramente cada aplicación.**

Por lo tanto, las empresas podrán proyectar nuevas instalaciones, reestructuraciones, adaptaciones o ampliaciones del layout con unos costes y tiempos de desarrollo acordes a sus posibilidades, que de otro modo serían incapaces de abordar.

Principales Aportaciones

Las principales aportaciones del presente trabajo son:

- Taxonomía de la reutilización del software.
- Criterios para el estudio del impacto que las metodologías de desarrollo de software tienen en la reutilización.
- Análisis del impacto de las arquitecturas genéricas de fabricación, de los métodos estructurados y de la tecnología orientada a objetos en la reutilización, así como de la Megaprogramación en la construcción del software.
- Metodología con reutilización sistemática para el desarrollo del software de control de FMSs. Se contribuye positivamente a esta área aportando:
  - Versión modificada del diseño en Y del Profesor Scheer para delimitar el ámbito del software de control de FMSs.
  - Versión modificada de la arquitectura propuesta por Robotiker para obtener una arquitectura genérica del dominio de tipo distribuido, que combina las características de los sistemas orientados a objetos y de los sistemas de eventos.
  - Especificación de mensajería del dominio.
  - Arquitecturas genéricas para los módulos identificados en la arquitectura genérica del dominio (Monitor, Mantenimiento on line, Scheduler dinámico, Control básico Dispatcher, gestores de recursos, servidores de dispositivos y Bus de mensajes).
  - Procesos explícitos para ejecutar las distintas etapas de desarrollo de un proyecto concreto.
- Criterios para medir el esfuerzo de adaptación de los módulos implementados en el dominio para obtener los particulares de una aplicación.

Las actividades realizadas en este trabajo han permitido definir varias líneas futuras de trabajo, que consisten en el desarrollo de:

- Herramientas para automatizar el análisis y diseño de un proyecto concreto.

- Generadores de código, que permitan la construcción de los distintos módulos identificados en la arquitectura del dominio.
- Repositorio basado en el modelo del dominio, con potentes procedimientos automatizados para la búsqueda, selección y reutilización de elementos.

Apéndice A

Glosario de términos para el dominio del software de control de FMSs

A

Arquitectura — Diseño de alto nivel que permite identificar los elementos que componen un sistema y las relaciones entre los mismos.

Atada — Grupo de operaciones de mecanizado que se pueden realizar sobre la pieza sin cambiar el amarre de la pieza en la máquina.

B

Bridge — Pareja de procesos que resuelven la comunicación entre dos aplicaciones que se ejecutan en diferentes plataformas.

Bus de mensajes — Software especializado en las tareas de mensajería entre procesos, que permite la comunicación y sincronización de los módulos identificados en la arquitectura genérica del dominio.

C

Carrusel — Sistema de almacenamiento de las herramientas en las máquinas.

Código Opitz — Método de codificación que describe los atributos de cada una de las piezas de una familia.

Control básico Dispatcher — Módulo software de la arquitectura genérica del dominio que se encarga de descomponer las tareas recibidas del Scheduler dinámico en servicios, secuenciando y coordinando su envío.

D

Diagrama de flujo de mensajes — Herramienta que permite determinar el comportamiento dinámico de un sistema, por medio de la identificación de los elementos involucrados y la representación del intercambio de mensajes entre dichos elementos.

E

Especificación de mensajería — Conjunto de servicios y eventos que intercambia un módulo, incluyendo sus descripciones detalladas y la secuencia del intercambio.

Evento — Mensaje que puede surgir como notificación de un servicio finalizado, o ante algún cambio de estado, situación de error o suceso.

F

Familia de piezas Conjunto de piezas que tienen en común un grupo de operaciones a realizar.

G

Garra Sistema de sujeción que utiliza un robot.

Gateway Pareja de procesos que resuelven la comunicación entre dos aplicaciones que se ejecutan en diferentes plataformas.

Gestor de recursos Módulo software de la arquitectura genérica del dominio que desempeña tareas especializadas en el tratamiento de recursos de difícil gestión.

L

Layout Distribución física de los dispositivos que componen la planta.

M

Mantenimiento on-line Módulo software de la arquitectura genérica del dominio que tiene a su cargo las tareas de mantenimiento y tratamiento de los recursos.

Middleware Soporte software que resuelve el problema de la comunicación entre procesos de forma transparente.

Modelo Abstracción que recoge la descripción de un sistema, problema, solución, etc.

Modelo de eventos Descripción detallada de los eventos.

Modelo de servicios Descripción detallada de las peticiones de servicio.

Monitor Módulo software de la arquitectura genérica del dominio que realiza la visualización del proceso en planta y proporciona la interface con el usuario.

Mordaza Sistema de sujeción de la pieza en la máquina para su mecanizado.

O

Operación Acción básica de fabricación a realizar sobre la pieza, que implica una transformación de la misma.

P

Palet	Dispositivo mecánico que desplaza a la pieza o herramienta sobre el sistema de transporte.
Paletizado	Operación de montaje del palet, los utillajes y la pieza o herramienta.
Petición de servicio	Mensaje que envían unos módulos a otros para solicitar la realización de un acción determinada o para pedir algún tipo de información.
Pieza	Elemento físico a fabricar a partir de la materia prima.
Plan de producción	Secuencia de operaciones de las piezas a fabricar en un sistema de fabricación, indicando con qué recurso y en qué instante.
Planificador off-line	Módulo responsable de la generación del plan de producción, que no tiene presente el estado real de la planta de fabricación.
Plataforma de integración	Soporte software que resuelve el problema de la comunicación entre procesos de forma transparente.

R

Ruta de fabricación	Secuencia ordenada de operaciones asignadas a máquinas en las cuales se puede llevar a cabo la fabricación de una pieza.

S

Scheduler dinámico	Módulo software de la arquitectura genérica del dominio que mantiene actualizado el plan de producción en respuesta a la situación real de la planta de fabricación.
Scheduler estático	Ver planificador off-line.
Servidor de dispositivo	Módulo software de la arquitectura genérica del dominio que controla y resuelve los aspectos específicos de los dispositivos físicos.
Sistema de control de FMSs	Software a cargo de la planificación, el control y el seguimiento de las actividades que se realizan en la planta de fabricación. También es responsable de la integración del propio sistema de control con las restantes funciones productivas de la empresa.

T

Tecnología de grupos — Clasificación en familia de piezas, aprovechando las similitudes de piezas y procesos para lograr mejores rendimientos en el diseño y la fabricación.

U

Utillaje — Elemento que se monta en el palet para fijar la pieza o herramienta.

W

Workstation — Área funcional que se encarga de una serie de tareas de fabricación, para lo cual tiene asignados determinados equipos de planta y sus controladores. Las workstations más habituales en un FMS de mecanizado por arranque de viruta son: workstation de paletizado, de transporte, de torno, de centro de mecanizado, y de despaletizado.

Apéndice B

Glosario de acrónimos para el dominio del software de control de FMSs

A

AGV Automated Guided Vehicle (vehículo guiado automáticamente).

API Application Program Interfaces (interfaces de programas de aplicación).

C

CACE Computer Aided Control Engineering (ingeniería de control asistida por ordenador)

CAD Computer Aided Design (diseño asistido por ordenador).

CAE Computer Aided Engineering (ingenieria asistida por ordenador).

CAM Computer Aided Manufacturing (fabricación asistida por ordenador).

CAMC Computer Aided Maintenance Control (control del mantenimiento asistido por ordenador).

CAPP Computer Aided Process Planning (planificación del proceso asistida por ordenador).

CAQ Computer Aided Quality Assurance (garantía de la calidad asistida por ordenador).

CBD Módulo Control básico Dispatcher.

CIM Computer Integrated Manufacturing (fabricación integrada por ordenador).

CN Control Numérico.

CNC Computer Numerical Control (control numérico computerizado).

D

DAE Distribution Automation Edition (edición para la automatización distribuida).

DAS Data Acquisition System (sistema de adquisición de datos).

DNC Direct Numerical Control (control numérico directo).

F

FMC Flexible Manufacturing Cell (célula de fabricación flexible).

FMEA Failure Mode and Effects Analysis (análisis de efectos y modos de fallo).

FMS Flexible Manufacturing System (sistema de fabricación flexible).

FTA Fault Tree Analysis (análisis del árbol de fallos).

G

GDNC Módulo gestor de programas de control numérico.

GHTAS Módulo gestor de herramientas.

GPAO Gestión de Producción Asistida por Ordenador.

GSI G2 Standard Interface (interface estándar para G2).

M

MFD Message Flow Diagram (diagrama de flujo de mensajes).

MNTO Módulo Mantenimiento on-line.

MRP Material Requirements Planning (planificación de requisitos de material).

O

OLE Object Linking and Embedding (comunicación y manipulación de objetos).

OPLAN Módulo servidor de operario.

P

PLC Programmable Logic Controller (controlador lógico programable).

PPC Production Planning and Control (control y planificación de la producción).

R

RC Robot Controller (controlador de robot).

RGV Rail Guided Vehicles (vehículos guiados por railes).

RIC Realtime Interface Co-Procesor (coprocesador para el interface en tiempo real).

RICES	RIC Extended Services (servicios extendidos para la RIC).
RPC	Remote Procedure Call (llamadas a procedimientos remotos).
S	
SCH	Módulo Scheduler dinámico.
SNCn	Módulo servidor de una máquina de control numérico.
SPLC	Módulo servidor de un PLC.
SRCn	Módulo servidor de un robot.
T	
TCP/IP	Transmission Control Protocol / Internet Protocol (protocolo para el control de transmisión/protocolo de red).

Acrónimos

A

ACD Activity Cycle Diagram (diagrama de ciclo de actividad).

ADFD Action Data Flow Diagram (diagrama de flujo de datos acciones).

AER Asociación Española de Robótica.

AFM Asociación española de Fabricantes de Máquinas-herramienta.

AMRF Advanced Manufacturing Research Facility (ayuda a la investigación de la fabricación avanzada).

ARIS ARchitecture for Information Systems (sistemas de información para arquitecturas).

ARS Análisis de Requisitos del Sistema.

ATP Automatización Tecnologías de la Producción.

C

CAI Computer Aided Industrie (industria asistida por ordenador).

CAO Computer Aided Organization (organización asistida por ordenador).

CASA Computer and Automated Systems Association (asociación de sistemas automatizados y computerizados).

CASE Computer Assisted Software Engineering (ingeniería del software asistida por ordenador).

CESA Computational Engineering in Systems Applications (Ingeniería computerizada en aplicaciones de sistemas).

CFD Control Flow Diagram (diagrama de flujo de control).

CFRP Conceptual Framework for Reuse Processes (marco conceptual para los procesos de reutilización).

CICYT Comisión Interministerial de Ciencia Y Tecnología.

CIM-OSA CIM Open System Architecture (arquitectura para sistemas abiertos CIM).

CIRP Institución internacional para la investigación de la ingeniería de producción.

CNMA Communications Network for Manufacturing Applications (redes de comunicación para aplicaciones de fabricación).

COPICS Communication-Oriented Production Information and Control System (sistema de control e información de la producción orientado a la comunicación).

CORBA Common Object Request Broker Architecture (arquitectura común de agentes de solicitud de objetos).

COSIMA COntrol Systems for Integrated MAnufacturing (sistemas de control para la fabricación integrada).

COTS Commercial-Off-The-Shelf (componentes comerciales reutilizables).

CSPEC Control SPECification (especificaciones de control).

D

DAAAM Danube-Adria Association for Automation and Metrology (Asociación del Danubio y Adriático para automatización y metrologia).

DCM1 Dispositivo centro de mecanizado Kondia B-500.

DCM2 Dispositivo centro de fresado Kondia K-76.

DCS Desarrollo de Componentes del Sistema.

DEC Diagrama de Estructura de Cuadros.

DFD Data Flow Diagram (diagrama de flujo de datos).

DoD Department of Defense (Departamento de defensa).

DPU Desarrollo de Procedimientos de Usuario.

DR1 Dispositivo robot Fanuc S-700.

DR2 Dispositivo robot Fanuc S-10.

DR3 Dispositivo robot Asea IRB6.

DSD Data Structure Diagram (diagrama de estructura de datos).

DTO1 Dispositivo torno CMZ TBI-450.

E

E/R Entity/Relationship diagram (diagrama entidad/relación).

EFS Especificación Funcional del Sistema.

EHU Euskal Herriko Unibertsitatea (Universidad del País Vasco).

ESC Electronic Systems Center (Centro de sistemas electrónicos).

ESI European Software Institute (Instituto europeo del software).

ESPRIT European Strategic Programme for Research and development in Information Technology (programa estratégico europeo para la investigación y el desarrollo en la tecnología de la información).

F

FAIM Flexible Automation and Intelligent Manufacturing (fabricación inteligente y automatización flexible).

FAM Factory Automation Model (modelo de automatización de factoría).

FBD Function Block Diagram (diagrama de bloques de función).

FC Factory Coordination (coordinación de factoría).

FIP Factory Instrumentation Protocol (protocolo de instrumentación de industria).

FS Function Subjects (áreas funcionales).

G

GEM Generic Equipment Model (modelo de equipamiento genérico).

GERAM Generalised Enterprise Reference Architecture and Methodology (metodología y arquitectura de referencia para una empresa genérica).

GIM GRAI Integrated Methodology (metodología integrada GRAI).

H

HOOMA Hierarchical and Object-Oriented Manufacturing systems Analysis (análisis jerárquico y orientado a objetos de sistemas de fabricación).

HVE Historia de la Vida de las Entidades.

I

ICAM International Computer Aerospace Manufacturing (fabricación internacional aerospacial por ordenador)

ICAM Integrated Computer-Aided Manufacturing (fabricación integrada asistida por ordenador) - IDEF0.

ICEIMT International Conference on Enterprise Integration Modeling Technology (Conferencia internacional sobre la tecnología de modelado de la integración de empresa).

IDEF0 Integration DEFinition language 0 (lenguaje de definición integrada 0).

IEM Integrated Enterprise Modelling (modelado de empresa integrado).

IFAC International Federation of Automatic Control (Federación internacional de control automático).

IFIP International Federation for Information Processing (Federación internacional para el procesamiento de la información).

IICE Information Integration for Concurrent Engineering (integración de la información para ingeniería concurrente).

IMS Intelligent Manufacturing System (sistema de fabricación inteligente).

IOD I/O Diagram (diagrama de entradas/salidas).

ISO International Standards Organization (Organización de estándares internacionales).

L

LDS Logical Data Structure (estructura lógica de datos).

M

MAP Manufacturing Automation Protocol (protocolo para la automatización de la fabricación).

MAP Ministerio para las administraciones públicas - MÉTRICA.

MFD Message Flow Diagram (diagrama de flujo de mensajes).

MMS Manufacturing Messaging Specification (especificación de mensajería de fabricación).

MSPEC Module SPECification (especificaciones de módulos).

N

NBS National Bureau of Standards (Buró nacional de estándares). Ver NIST.

NIST National Institute of Standards and Technology (Instituto nacional de estándares y tecnología).

O

OAM Object Access Model (modelo de acceso de objetos).

OCM Object Communication Model (modelo de comunicación de objetos).

OHMS Order Handling Manufacturing System (sistema de fabricación bajo pedido).

OL Object Lifecycles (ciclos de vida de los objetos).

OMG Object Management Group (grupo para la gestión de los objetos).

OMT Object Modeling Technique (técnica de modelado de objetos).

OOADA Object-Oriented Analysis and Design (análisis y diseño orientado a objetos).

OODLE Object-Oriented Design LanguagE (lenguaje de diseño orientado a objetos).

OOSA Object-Oriented Systems Analysis (análisis de sistemas orientado a objetos).

OOSE Object-Oriented Software Engineering (ingeniería del software orientada a objetos).

ORB Object Request Broker (agente de solicitud de objetos).

OSI Open System Interconnection (interconexión de sistemas abiertos).

P

PAC Production Activity Control (control de la actividad de producción).

PERA Purdue Enterprise Reference Architecture (arquitectura de referencia de empresa Purdue).

PLAS Product Line Asset Support Group (Grupo de asistencia a la línea de productos).

PLEC Product Line Engineering Centers (Centros de ingeniería de línea de productos).

PROFIBUS PROcess FieldBUS (bus de campo para proceso).

PSEPC	Process SPECification (especificación de procesos).
PTRn	Posiciones de acceso del sistema de transporte.
PYME	Pequeña Y Mediana Empresa.
R	
RCM	Reuse Capability Model (modelo de capacidad de la reutilización).
RDA	Relational Data Analysis (análisis relacional de datos).
REBOOT	REuse Based on Object-Oriented Techniques (técnicas orientadas a objetos basadas en la reutilización).
RMM	Reuse Maturity Model (modelo de madurez de la reutilización).
S	
SA/SD	Structured Analysis/Structured Design (análisis estructurado/diseño estructurado).
SADT	Strutured Analysis and Desig Technique (técnica de análisis y diseñi estructurado).
SAG	System Architectures Group (Grupo de arquitecturas de sistemas).
SC	Structure Chart (diagrama de estructura).
SCAI	Space Command and control Architectural Infraestructure
SEI	Software Engineering Institute (Instituto de ingenieria del software).
SEMI	Semiconductor Equipment and Material International (internacinal equipo y material semiconductor).
SER	Software Evolution and Reuse (reutilización y evolución del software).
SFC	Shop Floor Control (control de planta).
SFLC	Software-First Life Cycle (primer ciclo de vida del software).
SME	Society of Manufacturing Engineers (sociedad de ingenieros en fabricación).
SPC	Software Productivity Consortium (Consorcio para la productividad del software).
SPCSP	SPC Service Corporation (Corporación de servicios SPC).

SRD	Subsystem Relationship Diagram (diagrama de relación entre subsistemas).
SSADM	Structured Systems Analysis Method (método de análisis de sistemas estructurado).
STARS	Software Technology for Adaptable, Reliable Systems (tecnología software para sistema fiables y adaptables).
STD	State Transition Diagram (diagrama transición estado).
STP	State Process Table (tabla de procesos de estado).
STT	State Transition Table (tabla transición estado).
T	
TS2	Sistema de transporte TS-2.
U	
UML	Unified Modeling Language (lenguaje de modelado unificado).
UPV	Universidad del País Vasco.

Bibliografía

A

[Adiga, 93] Adiga, S., "Object-Oriented software for manufacturing systems". Chapman & Hall, 1.993.

[Alvarez, 95] Alvarez, E., "Una arquitectura de Scheduling integrada con Mantenimiento y Control de planta para células de fabricación flexible". Tesis Doctoral realizada en el Dpto. de Organización de Empresas de la UPV/EHU, 1.995.

[AMICE, 93] AMICE (Consorcio ESPRIT), "CIMOSA: Open System Architecture for CIM". Informe ESPRIT, Springer-Verlag, 1.993.

[Arzen, 92] Arzen, K.E., "GRAFCHART User Manual". Department of Automatic Control, Lund Technical University (Suecia), 1.992.

[Ayerbe, 96] Ayerbe, A., "A methodology for a maintenance system integrated with the control modules of an automated manufacturing system". Proceeding of the Sixth International FAIM Conference, Págs.: 591-600, 1996.

B

[Bakker, 91] Bakker, J.J.A., "DFMS: Architecture and Implementation of a Distributed Control System for FMS". Laboratory of Manufacturing Systems of the Delft University of Technology, 1.991.

[Bandinelli, 96] Bandinelli, S., Rementeria, S., "Software reuse introduction requires a process perspective". ESI (European Software Institute), 1.996.

[Barn, 97] Barn, W., Road, "Overview of SSADM". http://www.chl.co.uk, 1.997.

[Bauer, 94] Bauer, A., Bowden, R., Browne, J., Duggan, J, Lyons, G., "Shop-Floor Control Systems". Chapman & Hall, 1.991.

[Bernus, 94] Bernus, P., Nemes, L., "A Framework to Define a Generic Enterprise Reference Architecture and Methodology". http://www.cit.gu.edu.au/~bernus/taskforce/geram/report.v1/report/report.html, 1.994.

[Biggerstaff, 89] Biggerstaff, T.J., Perlis, A.J., "Software reusability. Concepts and models". ACM Press, Vol. I, 1.989.

[B-kin, 96] "B-Kin Links". B-Kin Software, 1.996.

[Booch, 94] Booch, G., "Object-oriented analysis and design with applications". Benjamin/Cummings Publishing Company, 1.994.

[Booch, 97] Booch, G., "Quality software and the Unified Modeling Language". http://www.rational.com/support/techpapers/soft_uml.html, 1.997.

[Bristow, 95] Bristow, D.J., Bulat, B.G., Burton, D.R., "Product line process development". STARS (Software Technology for Adaptable, Reliable Systems), 1.995.

[Burgos, 93] Burgos, A., López De Lacalle, L.N., Sánchez, J.A., Llorente, J.I., Sarachaga, M.I., "CIM: A case study". 4th International DAAAM Symposium, Págs.: 49-50, 1.993.

[Burgos, 94] Burgos, A., Llorente, J.I., López De Lacalle, L.N., Sánchez, J.A., Sarachaga, M.I., Martija, I., "CIM: A case study". Elektrotechnik und informationstechnik, Nº 6, Págs.: 331-337, 1.994.

[Burgos, 94a] Burgos, A., Sarachaga, M.I., Llorente, J.I., López, J.M., Pereda, O., Santamaria, J.M., "El papel de las plataformas de integración en entornos de fabricación". Anales de Ingeniería Mecánica, Vol. 2, Págs.: 103-110, 1.994.

[Burgos, 95] Burgos, A., Sarachaga, M.I., Llorente, J.I., López, J.M., "The role of integration platforms". 1st World Congress on Intelligent Manufacturing Process & Systems, Vol. 1, Págs.: 401-410, 1.995.

[Burgos, 96] Burgos, A., Llorente, J.I., Sarachaga, M.I., López, J.M., "Flujo de información entre aplicaciones independientes en entornos de fabricación". Anales de Ingeniería Mecánica, Vol. 2, Págs.: 487-496 , 1.996.

[Burgos, 97] Burgos, A., "La integración de los sistemas de fabricación: software de comunicaciones entre aplicaciones y dispositivos". Tesis Doctoral realizada en el Dpto. de Ingeniería Mecánica de la UPV/EHU, 1.997.

[Buschmann, 92] Buschmann, F., "Rational architectures for object-oriented software systems". Proceeding of the Fifth International Conference Software Engineering & its applications, Págs.: 379-390, 1.992.

C

[CACI, 93] "Simfactory II and Simprocess for Windows". CACI Products Company, 1.993.

[CASA/SME, 93] CASA/SME (Computer and Automated Systems Association/ Society of Manufacturing Engineers), "The New Manufacturing Enterprise Wheel". http://www.sme.org/memb/casa/wheel.html, 1.993.

[Chen, 96] Chen, Y., "A Practicable Scheme of CIM System Architecture". Proceeding of the Sixth International FAIM Conference, Págs.: 621-630, 1.996.

[CICYT, 94] "Integración de técnicas avanzadas de mantenimiento y control para la mejora de la disponibilidad de los sistemas de fabricación. Acción especial". Ref.: TAP94-1435-E. Dpto. de Ingeniería Mecánica de la UPV/EHU, 1.994.

[CICYT, 95] "Integración de técnicas avanzadas de mantenimiento y control para la mejora de la disponibilidad de los sistemas de fabricación.". Ref.: TAP95-0977-C02-01. Dpto. de Ingeniería Mecánica de la UPV/EHU, 1.995-97.

[CIMOSA, 96] Asociación CIMOSA, "CIMOSA. A Primer on key concepts, purpose and business value". http://cimosa.cnt.pl/Docs/Primer/primer0.htm, 1.996.

[Cohen, 95] Cohen, S., Friedman, S., Martin, L., Solderitsch, N., Webster, R., "Product line identification for ESC-Hanscom". SEI (Software Engineering Institute), 1.995.

[Cohen, 96] Cohen, S., Friedman, S., Martin, L., Royer, T., Solderitsch, N., Webster, R., "Concept of operations for ESC product line approach". SEI, 1.996.

[Coleman, 94] Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., Jeremaes, P., "Object-oriented develoment. The Fusion method". Prentice-Hall, 1.994.

[Conn, 97] Conn, R., "Software reuse course (Version 3)". Software Engineering Department, Monmouth University (New Jersey), 1.997.

[Corriveau, 96] Corriveau, J.P., "Traceability process for large object-oriented projects". Computer, Vol. 29, Nº 9, Págs.: 63-68, 1.996.

D

[DeMarco, 79] DeMarco, T., "Structured analysis and system specification". Prentice Hall, 1.979.

[Deregibus, 91] Deregibus, F., Bobbio, M., Rusina, F., "Open Systems and Manufacturing Software Integration Platforms". Proceedings of the Seventh CIM-Europe Annual Conference, Págs.: 33-43, 1.991.

F

[Fabian, 94] Fabian, M., Lennartson, B., "Petri Nets and control synthesis: an object-oriented approach". Proceedings of IMS'94. 1.994.

[FACT, 97] FACT, "Shop Floor Control in part manufacturing and assembly". http://www.pt.wb.utwente.nl/projects/shopfloor, 1.997.

[Freemon, 92] Freemon, B.W., Crispen, R.G., "Testing a technology for reuse". The Boeing Company, 1.992.

G

[Gensym, 92] "G2 Reference Manual". Gensym Corporation, 1.992.

[Gerhart, 94] Gerhart, S., "Software reuse initiative. Technology Roadmap (Version 1.2). DoD (Department of Defense), 1.994.

[Goenaga, 91] Goenaga, J.M., Zugasti, I., "Metodologías: de la Información a la Producción". Automatización, Nº 37, Págs.: 32-44, 1.991.

[Gómez, 94] Gómez, V., Burgos, A., Sarachaga, M.I., Diez, L., "Estructura de comunicaciones industriales en entornos CIM". Anales de Ingeniería Mecánica, Vol. 2, Págs.: 111-119, 1.994.

[Gong, 96] Gong, D.Ch., "A structured approach for designing shop floor control information systems". Proceeding of the Sixth International FAIM Conference, Págs.: 705-712, 1.996.

H

[HyperLogic, 92] "CubiCalc. The third wave in intelligent software". HyperLogic Corporation, 1.992.

[HockWare, 95] "Vispro/C for OS/2. User Manual". HockWare, 1.995.

I

[IBM, 89] "RIC Extended Services user's guide". Communication support. IBM, 1.989.

[IBM, 91] "Distributed Automation Edition – Communication System/2. Technical guide and reference". Plant Floor Series IBM, 1.991.

[IBM, 92] "Plantworks. Reference guide". Plant Floor Series IBM, 1.992.

[IDEF, 97] "The U.S. Air Force IDEF methods. A structured approach to enterprise modeling and analysis". http://www.idef.com/default.htm, 1.997.

[IDEF0, 93] "Announcing the standard for Integration DEFinition for function modeling (IDEF0)". Federal Information Processing Standards Publication 183, 1.993.

[IDEF1X, 93] "Announcing the standard for Integration DEFinition for information modeling (IDEF1X)". Federal Information Processing Standards Publication 184, 1.993.

[IDEF4, 95] "Information Integration for Concurrent Engineering (IICE). IDEF4 object-oriented design method report (Version 2)". Knowledge Based Systems Inc., 1.995.

J

[Jacobson, 92] Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G., "Object-oriented software engineering. A use case driven approach". Addison-Wesley, 1.992.

[Joshi, 94] Joshi, S.B., Smith, J.S., "Computer Control of Flexible Manufacturing Systems. Research and Development". Chapman & Hall, 1.994.

K

[Karlsson, 95] Karlsson, E.A., "Software Reuse. A Holistic Approach". John Wiley & Sons, 1.995.

[Koch, 95] Koch, Th., Week, M., "COSMOS – An open architecture for discrete cell control". Proceedings of the 27th CIRP International Seminar on Manufacturing Systems. Design, Control and Analysis of Manufacturing Systems, Págs.: 18-26, 1.995.

[Kosanke, 97] Kosanke, K., "Enterprise Integration and Standardization". ICEIMT'97 (International Conference on Enterprise Integration Modeling Technology), Workshop 2 (USA, 21-23 de Abril), 1.997.

[Krueger, 92] Krueger, C.W., "Software reuse". ACM Computing Surveys, Vol. 24, Nº 2, Págs.: 131-183, 1.992.

[Kusiak, 92] Kusiak, A., "Intelligent Design and Manufacturing". John Wiley & Sons, 1.992.

L

[Lanner, 96] "Witness". Lanner Group Ltd., 1.996.

[Llorente, 93] Llorente, J.I., Sánchez, J.A., López De Lacalle, L.N., Burgos, A., Sarachaga, M.I., "Advanced control for automated workshops". 4th International DAAAM Symposium, Págs.: 179-180, 1.993.

[Llorente, 94] Llorente, J.I., López De Lacalle, L.N., Sánchez, J.A., Sarachaga, M.I., Martija, I., Burgos, A., "Advanced control for automated plants". Elektrotechnik und informationstechnik, Nº 6, Págs.: 266-269, 1.994.

[Llorente, 94a] Llorente, J.I., Burgos, A., Sarachaga, M.I., Sánchez, J.A., López De Lacalle, L.N., "Open architecture for the control of automated plants". 5th International DAAAM Symposium, Págs.: 247-248, 1.994.

[Llorente, 94b] Llorente, J.I., Sánchez, J.A., Burgos, A., Sarachaga, M.I., López De Lacalle, L.N., "Integration of design and tool management in an advanced manufacturing system". 5th International DAAAM Symposium, Págs.: 249-250, 1.994.

[Llorente, 94c] Llorente, J.I., López De Lacalle, L.N., López, J.M., Sánchez, J.A., Burgos, A., Sarachaga, M.I., "Planta piloto de integración en técnicas de fabricación". IMHE, Nº 200, Págs.: 44-50, 1.994.

[Llorente, 96] Llorente, J.I., Burgos, A., López, J.M., Sarachaga, M.I., "Open architecture for control of automated plants". Manufacturing Systems, Vol. 25, Nº 1, Págs.: 29-36, 1.996.

[Llorente, 96a] Llorente, J.I., López, J.M., Sarachaga, M.I., Burgos, A., "Experiencias en la aplicación de técnicas de ingeniería del software al control de sistemas de fabricación". XI Congreso de Máquinas Herramienta y Tecnologías de Fabricación, Sesión 6, Págs.: 1-16, 1.996.

[Llorente, 97] Llorente, J.I., Sarachaga, M.I., Burgos, A., Viñolas, J., Bueno, R., "Reuse of control software for manufacturing systems". Annals of the CIRP, Vol. 46, Nº 1, Págs.: 403-406, 1.997.

[López, 94] López, J.M., Llorente, J.I., Santamaria, J.M., Pereda, O., Alvarez, E., Burgos, A., Sarachaga, M.I., "Métodos gráficos para el control de sistemas de fabricación". Anales de Ingeniería Mecánica, Vol. 2, Págs.: 121-128, 1.994.

[López, 95] López, J.M., "Metodología para el desarrollo de software de control de sistemas de fabricación". Tesis Doctoral realizada en el Dpto. de Ingeniería Mecánica de la UPV/EHU, 1.995.

[López, 95a] López, J.M., Santamaria, J.M., Llorente, J.I., Sarachaga, M.I., "Cell control design and implementation using graphical tools". 1st World Congress on Intelligent Manufacturing Process & Systems, Págs.: 392-400, 1.995.

[Lynggaard, 96] Lynggaard, H.J., Alting, L., "Cell Control Engineering". Proceeding of the Sixth International FAIM Conference, Págs.: 178-187, 1996.

M

[Maglica, 97] Maglica, R., "Improving the PAC shop-floor control architecture to better support implementation". Computers in Industry (Elsevier), Vol. 33, Págs.: 317-322, 1.997.

[Marciniak, 94] Marciniak, J.J., "Encyclopedia of Software Engineering". John Wiley & Sons, Vol. 1 y 2, 1.994.

[Mayer, 95] Mayer, R.J., Menzel, C.P., Painter, M.K., Witte, P.S., Blinn, T., Perakath, B., "Information Integration for Concurrent Engineering (IICE). IDEF3 Process Description Capture Method report". Knowledge Based Systems (University Drive East), 1.995.

[MÉTRICA, 95a] "MÉTRICA v.2.1. Metodología de planificación y desarrollo de sistemas de información. Guía de referencia". MAP (Ministerio para las Administraciones Públicas), 1.995.

[MÉTRICA, 95b] "MÉTRICA v.2.1. Metodología de planificación y desarrollo de sistemas de información. Guía de técnicas". MAP, 1.995.

[Meyer, 91] Meyer, W., "Expert Systems in Factory Management: Knowledge-Based CIM". Ellis Horwood Books in Information Technology. John Wiley & Sons, 1.991.

[Microsoft, 96] "Microsoft Access para Windows 95. Paso a paso". Microsoft Press, McGraw Hill, 1.996.

[Mowbray, 95] Mowbray, T.J., Zahavi, R., "The essential CORBA. Systems integration using distributed objects", Object Management Group. John Wiley & Sons, 1995.

[Mulder, 97] Mulder, M.J., Koorn, M.P., van Aken, W., "SADT". http://wwwis.cs.utwente.nl:8080/dmrg/MEE97/misop001/public_html 1.997.

O

[O'Connor, 94] O'Connor, J., Mansour, C., Turner-Harris, J., Campbell, G.H., "Exploring systematic reuse for command and control systems". SPCSC (Software Productivity Consortium Service Corporation), 1.994.

[Object, 97] "ObjectStore de Object Design". http://www.odi.com, 1.997.

[Objectory, 96] "Objectory". http://www.rational.com/pst/products/obj_broch.html, 1.996.

[OMG, 97] "OMG adopts UML and meta object facility specifications". http://www.rational.com/world/press/releases/data/omgaccept.html, 1.997.

P

[Paci, 96] Paci, M., Hallsteinsen, S., "SER (Software Evolution and Reuse). Experience book". Proyecto ESPRIT 9809, 1.996.

[Potier, 97] Potier, D., "Managing top-down reuse". Technical Journal (suplemento de Syseca Magazine), N° 15, Págs.: 1, 1.997.

[Pressman, 93] Pressman, R.S., "Ingenieria del Software. Un Enfoque Práctico". McGraw Hill, 1.993.

Q

[Quinn, 96] Quinn, B., Shute, D., "Windows sockets network programming". Addison Wesley, 1.996.

R

[Ranky, 90] Ranky, P.G., "Flexible Manufacturing Cells and Systems in CIM". CIMware Ltd., 1.990.

[Rational, 92] "Rational Rose for Windows User Reference Guide". Rational, 1.992.

[Rational, 97] "Rational Objectory process (Version 4.1). Your UML process". http://www.rational.com/support/techpapers/toratobjprcs/, 1.997.

[Rembold, 93] Rembold, U., Nnaji, B.O., Storr, A., "Computer Integrated Manufacturing and Engineering". Addison Wesley, 1.993.

[Robotiker, 92] Robotiker, "Propuesta de arquitectura para sistemas de control de células de fabricación flexible". Proyecto 103/92. Departamento de Sistemas de Control Industrial, 1.992.

[Rumbaugh, 94] Rumbaugh, J., "OMT papers. The OMT summary introduction". http://www.rational.com/support/techpapers/omt/summary.html, 1.994.

[Rumbaugh, 96] Rumbaugh, J., Blaha, M., Premerlan, W., Eddy, F., Lorensen, W., "Modelado y diseño orientados a objetos. Metodologia OMT". Prentice Hall, 1.996.

S

[Sarachaga, 97] Sarachaga, M.I., Kahoraho, E., Burgos, A., Llorente, J.I., "Metodología con reutilización sistemática en el desarrollo del software de control para FMS". 5º Congreso AER-ATP: Las nuevas fronteras de la automatización, Págs.: 361-372, 1.997.

[Sarachaga, 98] Sarachaga, M.I., Llorente, J.I., Burgos, A., "How software development methodology affects the openness of a manufacturing system". CESA'98, Págs.: 306-311(CD-Rom), 1.998.

[Scheer, 91] Scheer, A.W., Hars, A., "Enterprise-Wide data modelling. The basis for integration". Proceedings of the Seventh CIM-Europe Annual Conference, Págs.: 331-344, 1.991.

[SFLC, 90] "Software-First Life Cycle. Final definition". IBM System Integration Division, 1.990.

[Shlaer, 88] Shlaer, S., Mellor, S.J., "Object-Oriented Systems Analysis. Modeling the world in data". Prentice Hall, 1.988.

[Shlaer, 92] Shlaer, S., Mellor, S.J., "Object Lifecycles. Modeling the world in states". Prentice Hall, 1.992.

[Smith, 92] Smith, J.S., Joshi, S.B., "Reusable Software Concepts Applied to the Development of FMS Control Software". Computer Integrated Manufacturing, Vol. 5, Nº 3, Págs.: 182-196, 1992.

[Smith, 94] Smith, J.S., Joshi, S.B., "A Shop Floor Controller Class for Computer Integrated Manufacturing". http://tamcam.tamu.edu/pubs/sfcclass.htm, 1.994.

[SPC, 93] SPC (Software Productivity Consortium), "Overview of Megaprogramming course (Version 01.00.02)". http://www.software.org/pub/Products/megap_orig_may_96/htmls/0101.htm, 1.993.

[SPC, 93a] SPC, "Reuse-driven software processes guidebook (Version 02.00.03)". SPCSC (SPC Service Corporation), 1.993.

[SPC, 93b] SPC, "Reuse adoption guidebook (Version 02.00.03)". SPCSC (SPC Service Corporation), 1.993.

[STARS, 93] STARS (Software Technology for Adaptable, Reliable Systems), "Application Engineering with domain-specific reuse. Volume I: instructor's guide". Paramax Systems Corporation, 1.993.

[STARS, 93a] STARS, "STARS Conceptual framework for reuse processes. Volumen I: Definition". Unisys Corporation, 1.993.

[STARS, 96] STARS, "Domain engineering methods and tools handbook. Volumen I: Methods". Loral Defense Systems-East, 1.996.

[Sugimura, 95] Sugimura, N., Moriwaki, T., Hozumi, K., Shinohara, Y., "Modeling of Holonic Manufacturing System and Its Application to Real Time Scheduling". Proceedings of the 27th CIRP International Seminar on Manufacturing Systems. Design, Control and Analysis of Manufacturing Systems, Págs.: 401-410, 1.995.

T

[Toh, 97] Toh, K.T.K., Newman S.T., Bell, R., "The capture of human interactions in Enterprise Modelling". ICEIMT'97 (International Conference on Enterprise Integration Modeling Technology), 1.997.

[Tönshoff, 95] Tönshoff, H.K., Winkler, M., "Shop Control for Holonic Manufacturing System". Proceedings of the 27th CIRP International Seminar on Manufacturing Systems. Design, Control and Analysis of Manufacturing Systems, Págs.: 329-336, 1.995.

[Topper, 94] Topper, A., Ouellette, D., Jorgensen,P., "Structured methods. Merging models, techniques, and case". McGraw Hill, 1.994.

U

[UML, 97] "UML summary". http://www.rational.com/uml/html/summary/, 1.997.

V

[Vidal, 97] Vidal, M., "MEMO MERISE. Qu'est-ce que MERISE?". http://www.guetali.fr/home/vidalmat , 1.997.

[VOICE II, 95] "Validating OSA in Industrial CIM Environments (CIM-OSA)". ESPRIT 6682, http://www.prosoma.lu/cgi-bin/bsearch.py, 1.995.

W

[Waldner, 92] Waldner, J.B., "CIM. Principles of Computer Integrated Manufacturing". John Wiley & Sons, 1.992.

[Wei, 96] Wei, J., Zhou, Ch., "Cell control rapid prototyping using object oriented technology and SEMI standards". Proceeding of the Sixth International FAIM Conference, Págs.: 263-272, 1996.

[Williams, 97] Williams, T., "Modeling and Architecture (TC184 SC5 WG1)". http://www.mel.nist.gov/div826/subject/sc5/wg1/ger70429.htm, 1.997.

[Wu, 92] Wu, B., "Manufacturing systems design and analysis". Chapman & Hall, 1.992.

[Wu, 94] Wu, B., "Manufacturing systems design and analysis. Context and techniques". Chapman & Hall, 1.994.

Y

[Yourdon, 79] Yourdon, E., Constantine, L.L., "Structured design: fundamentals of a discipline of computer program and systems design". Prentice Hall, 1.979.