

Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos.

Itinerario Ingeniería de Software (Código 31105151)

Generación de código a partir de estructuras
jerárquicas de procesos elementales.
Aplicación a SAP ERP.

Autor: José Antonio Reguera Díaz

Director: Ismael Abad Cardiel.

Curso: 2016 / 2017.

Convocatoria: Junio 2017.

Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos.

Itinerario Ingeniería de Software (Código 31105151)

Generación de código a partir de estructuras
jerárquicas de procesos elementales.
Aplicación a SAP ERP.

Autor: José Antonio Reguera Díaz
Director: Ismael Abad Cardiel.

**DECLARACIÓN JURADA DE AUTORÍA DEL TRABAJO
CIENTÍFICO, PARA LA DEFENSA DEL TRABAJO FIN DE
MASTER**

Fecha: 11/06/2017.

Quién suscribe:

Autor(a): José Antonio Reguera Díaz D.N.I/N.I.E/Pasaporte.: 31638708T
--

Hace constar que es la autor(a) del trabajo:

Generación de código a partir de estructuras jerárquicas de procesos elementales. Aplicación a SAP ERP.
--

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores, se han referenciado debidamente en el texto de dicho trabajo.

DECLARACIÓN:

- ✓ Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente por otros autores, en soporte papel ni en formato digital.
- ✓ Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.
- ✓ No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.

Fdo.





IMPRESO TFD05_AUTOR
AUTORIZACIÓN DE PUBLICACIÓN
CON FINES ACADÉMICOS



**Impreso TFD05_Autor. Autorización de publicación
y difusión del TFD para fines académicos**

Autorización

Autorizo/amos a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del/los Autor/es

Juan del Rosal, 16
28040, Madrid

Tel: 91 398 89 10
Fax: 91 398 89 09

www.issi.uned.es

RESUMEN

El objetivo de este proyecto es construir un generador que use como entrada una base de datos con una estructura jerárquica de procesos elementales y su aplicación en un entorno SAP ERP. Vamos a investigar la posibilidad de tener una base de datos con código jerarquizado de la forma de una estructura de producto (Bill of Materials) inspirada en los procesos de Ingeniería de Configuración de materiales de la industria manufacturera.

Así mismo, esta base de datos pretende tener una doble función. Por un lado la de facilitar la reutilización de código y por otro el de ser una base de conocimiento “knowledge base” en donde tener almacenados los procesos elementales que nos servirán para construir las aplicaciones para dar cobertura a las reglas del negocio basadas en el ERP pero con menor coste y tiempo de desarrollo.

Con ello pretendemos dar respuesta entre otras cosas a la necesidad de automatización de la generación de informes que se produce tras la implantación de un sistema ERP pues de esta manera descompondremos los programas en procesos que podremos depurar individualmente acortando el tiempo de desarrollo y aumentando y sobre todo fomentando la reusabilidad.

Como SAP ERP es el líder en el segmento, vamos a estudiar su aplicación e implantación en un entorno SAP ERP como alternativa a desarrollo e ABAP. Estudiaremos las herramientas que nos provee SAP para tal fin y las formas de conexión de SAP con un programa externo.

Para ello un informe o programa lo vamos a asimilar como un árbol de procesos. Cada proceso estará clasificado con un tipo y tendrá una serie de relaciones padre – hijo. Las relaciones padre – hijo estarán sometidas a criterios de efectividad con lo que estamos hablando de listas configurables. La secuencia de procesos a ejecutar se obtendrá desde las relaciones padre – hijo mediante un algoritmo recursivo. Unas plantillas eRB por cada tipo de proceso servirán para construir el programa objetivo.

Este trabajo constará de dos programas JRuby y una base de datos que compartirán ambos programas. En la base de datos informaremos de los programas y sus procesos elementales jerarquizados y con sus criterios de efectividad. El programa **Desarrollador** por un lado nos servirá para desarrollar y depurar cada uno de los procesos independientes hasta construir la solución y una vez construida nos permitirá ejecutarla a partir de la información contenida en la base de datos y otro **Generador** que nos construirá el programa objetivo a partir de la información de la base de datos de tal forma que yo no necesitaré de la base de datos con la estructura para su ejecución.

Lista de palabras clave

ERP, Lista de materiales, Business intelligence, Implementation outcome, Arquitectura cliente – servidor, ABAP, Arbol de producto, Remote Function Call, OpenSQL, Middleware, Generador.

EXECUTIVE SUMMARY

The goal of this project is to build a generator that uses as input a database with a hierarchical structure of elementary processes and, later on, its application in an SAP ERP environment.

The construction of this database has been inspired in the hierarchical data structures of the Bills of Materials used in the manufacturing industry.

This database will have a dual role. On the one hand to facilitate the reuse of code and on the other to be a “knowledge base” where having stored the elementary processes that will serve to build applications to cover business rules based on the ERP.

With this we intend to react among other things to the need for automation of reporting that happens after the implementation of an ERP system because in this way we decompose the programs into processes that can be individually debugged, diminishing development time and increasing and encouraging reusability.

INDICE

Calificaciones.....	2
RESUMEN	5
Lista de palabras clave	5
EXECUTIVE SUMMARY	6
INDICE	7
LISTA DE FIGURAS Y TABLAS.....	9
1. Introducción	10
1.1 ¿Qué es un sistema ERP? [5]	10
1.2 Problemas con y tras la implantación de un sistema ERP.....	10
1.3 Introducción a SAP ERP	11
1.4 Arquitectura de SAP ERP	12
1.5 ABAP [7]	13
1.6 Diccionario de SAP [4]	14
1.7 Métodos modernos de control de producción [6].....	15
2. Planteamiento del problema.....	17
2.1 Contexto del problema.....	17
2.2 Objetivos concretos.....	18
3. Estado de la cuestión	19
3.1 Limitaciones de ABAP y Open SQL.....	19
3.2 Herramientas para hacer informes en SAP.....	20
SQVI QuickViewer	20
ABAP/Query	22
BI Tools.....	22
3.3 Funciones (RFC) que acceden a la base de datos	23
RFC_READ_TABLE.....	23
RFC_READ_TABLE ampliados	24
Demás rfc enabled function modules y BAPIs	25
3.4 Middleware sobre el que ejecutar RFC.....	25
MS OLE	26
Java.....	26
Web Services	26
4. Resolución	27
4.1 Lenguaje, base de datos y middleware	27

JRuby [9][10][11][12][13][14][15]	27
SQLite3 [16]	27
SEQUEL [17]	27
SAP JCo3 [18]	28
4.2 Descripción del Programa a generar	28
4.3 Descripción de la solución	28
4.4 Base de datos GenDB	29
4.5 Tipos de proceso	31
4.6 Proceso sapsql	32
4.7 Programa Desarrollador	32
4.8 Programa Generador	35
4.7 Programa Generado	36
4.10 Z_RFC_SQL	36
5 Casos de Estudio	37
5.1 Caso de estudio sin SAP	37
5.2 Caso de estudio con SAP	44
5.2.1 MaterialesSAP	45
5.2.2 PreplaMCSAP	46
5.2.3 PreplaDDWSAP	47
6. Conclusiones y líneas futuras	51
Apéndices	52
A1 – Programa Desarrollador	52
A2 – Programa Generador	60
A3 – Plantillas del generador	63
A4 – Salida.rb Caso 2	68
Referencias	72
Siglas	73

LISTA DE FIGURAS Y TABLAS

Figura 1: Resultados de la Implantación de ERP.....	11
Figura 2: Proveedores ERP más frecuentemente seleccionados como candidato.	12
Figura 3: Arquitectura de un sistema SAP ERP.	13
Figura 4: Arbol de producto.	15
Figura 5: Sistema de transporte entre entornos SAP ERP	19
Figura 6: Transacción SQVI; imagen inicial	21
Figura 7: Transacción SQVI; Entorno gráfico de unión de tablas	21
Figura 8: Transacción SQVI; Salida de la transacción.....	22
Figura 9: Imagen inicial del editor de funciones ABAP (SE37)	23
Figura 10: Pestaña Atributos (SE37)	23
Figura 11: Parámetros import RFC_READ_TABLE.....	24
Figura 12: Parámetros table RFC_READ_TABLE	24
Figura 13: Parámetros import Z RFC_READ_TABLE2.....	25
Figura 14: Parámetros table BAPI_PRODORD_GET_LIST.....	25
Figura 15: Script VBA para conectar a SAP via SAP.Functions OLE Object.....	26
Figura 16: Script Ruby para declarar clases para Java Connector 3.0.....	26
Figura 17: Diagrama del proceso de generación.	29
Figura 18: Relaciones de las tablas de la base de datos GenData.sqlite3.....	30
Tabla 19: Esquema de la tabla progmsr	30
Tabla 20: Esquema de la tabla itemmsr	30
Tabla 21: Esquema de la tabla prodstru	31
Tabla 22: Tabla con los tipos de proceso.....	31
Figura 23: Clase Desarrollador.	33
Figura 24: Clase Generador.	35
Figura 25: Clase Generada.....	36
Figura 26: Estructura de procesos elementales de numeros_00.....	38
Figura 27: Descripción funcional de numeros_00	38
Figura 28: Estructura de procesos elementales de numeros_01.....	39
Figura 29: Descripción funcional de numeros_01	39
Figura 30: Estructura de procesos elementales de numeros_02.....	40
Figura 31: Descripción funcional de numeros_02	40
Figura 32: Estructura de procesos elementales de numeros_03.....	41
Figura 33: Descripción funcional de numeros_03	41
Figura 34: Relación entre los procesos elementales de todos los programas.	42
Tabla 35: Contenido de la tabla progmsr del ejemplo 1.	42
Tabla 36: Contenido de la tabla itemmsr del ejemplo 1.	42
Tabla 37: Contenido de la tabla prodstru del ejemplo 1.	43
Figura 38: Estructura de procesos elementales de MaterialesSAP.....	45
Figura 39: Estructura de procesos elementales de PreplaMCSAP	46
Figura 40: Estructura de procesos elementales de PreplaDDWSAP	48
Figura 41: Flujo en la base de datos del ejemplo 2.....	49
Tabla 42: Contenido de la tabla progmsr del ejemplo 2.	49
Tabla 43: Contenido de la tabla prodstru del ejemplo 2.	49
Tabla 44: Contenido de la tabla itemmsr del ejemplo 2.	50

1. Introducción

1.1 ¿Qué es un sistema ERP? [5]

Los sistemas de planificación de recursos empresariales ('ERP', por sus siglas en inglés, enterprise resource planning) son los sistemas de información gerenciales que integran y manejan muchos de los negocios asociados con las operaciones de producción y de los aspectos de distribución de una compañía en la producción de bienes o servicios.

La planificación de recursos empresariales es un término derivado de la planificación de recursos de manufactura (MRPII) y seguido de la planificación de requerimientos de material (MRP); sin embargo los ERP han evolucionado hacia modelos de suscripción por el uso del servicio (SaaS, cloud computing).

Los sistemas ERP típicamente manejan la producción, logística, distribución, inventario, envíos, facturas y contabilidad de la compañía de forma modular. Sin embargo, la planificación de recursos empresariales o el software ERP pueden intervenir en el control de muchas actividades de negocios como ventas, entregas, pagos, producción, administración de inventarios, calidad de administración y la administración de recursos humanos.

Los sistemas ERP son llamados ocasionalmente back office (trastienda) ya que indican que el cliente y el público general no tienen acceso a él; asimismo, es un sistema que trata directamente con los proveedores, no estableciendo únicamente una administración de la relación con ellos (SRM). No obstante progresivamente el software ERP tiende a ocupar todos los espacios de la organización, absorbiendo las funciones del CRM. De hecho las principales compañías productoras de CRM del mundo fueron absorbidas por empresas de software ERP en los últimos diez años.

Los ERP funcionaban ampliamente en las empresas. Entre sus módulos más comunes se encuentran el de manufactura o producción, almacenamiento, logística e información tecnológica, incluyen además la contabilidad, y suelen incluir un sistema de administración de recursos humanos, y herramientas de mercadotecnia y administración estratégica.

1.2 Problemas con y tras la implantación de un sistema ERP

La implantación de un sistema ERP no siempre se concluye con éxito. El ya de por sí largo plazo de implantación se suele ver sobrepasado así tanto como el presupuesto de implantación. Además en un alto porcentaje, los ejecutivos de las empresas en que se implanta tienen una sensación indiferente o incluso de fracaso tras el proceso de implantación.

Tras la implantación del sistema ERP y como consecuencia de no dar respuesta a determinadas reglas del negocio, es normal la aparición de sistemas de información paralelos basados en herramientas ofimáticas que a la larga más que una ayuda son un riesgo para la ejecución del negocio de la compañía que los implanta. Así mismo es normal ver que el ERP no da respuesta a las necesidades de generación de informes de la compañía, con lo que en vez de disminuir, aumenta la carga del trabajo administrativo de la empresa.

Una solución a los problemas de generación de informes son los sistema BI o de inteligencia empresarial (business intelligence). Pero estos sistemas son otro desembolso importante para la empresa y no suelen contener toda la información que contiene el ERP, sino que necesitan

de unos procesos de carga o sincronización con unas pautas de tiempo determinadas.

Implementation Outcome

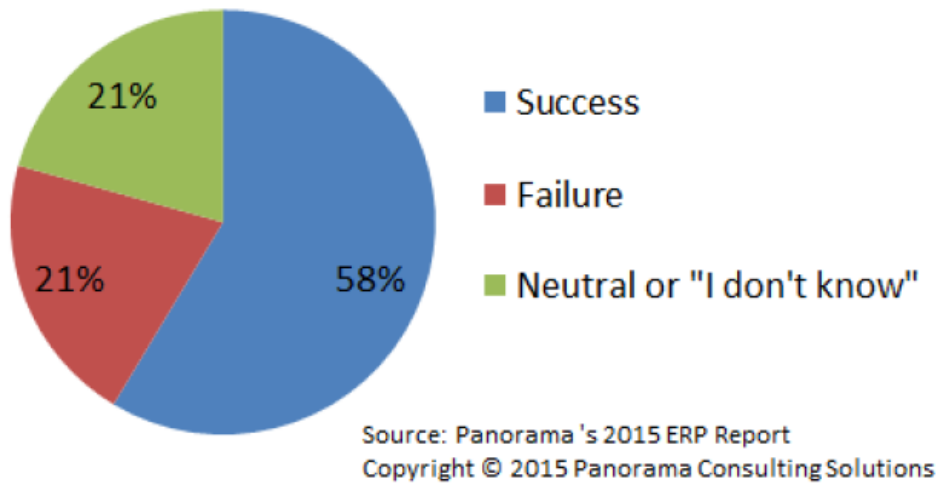
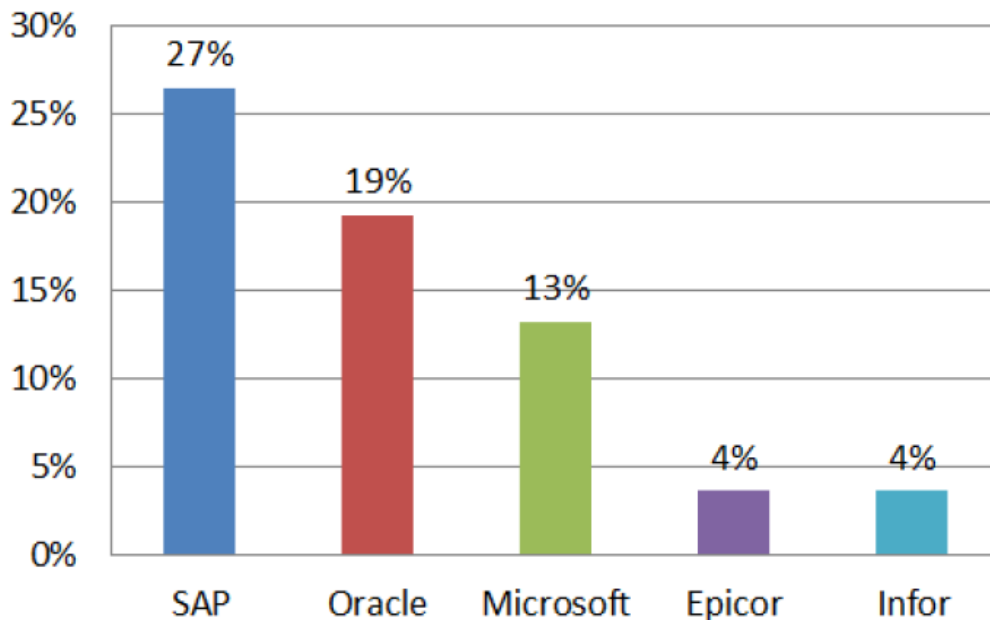


Figura 1: Resultados de la Implantación de ERP

1.3 Introducción a SAP ERP

SAP ERP se ha convertido en un estándar "de facto" debido a su amplia implantación. De acuerdo a fuentes de SAP [1] el 87% de las empresas que están en la lista Forbes Global2000 [2] de las compañías más grandes del mundo son clientes de SAP. En un estudio más global realizado por "Panorama Consulting" en 2015 (2015 ERP Report) [3] SAP es la compañía que ha sido seleccionada para aparecer como candidata a la implantación de ERP más veces con un 27% y a su vez la que ha sido más veces seleccionada para la implantación con un 39%.

Most Frequently Short-Listed Vendors



Source: Panorama 's 2015 ERP Report
Copyright © 2015 Panorama Consulting Solutions

Figura 2: Proveedores ERP más frecuentemente seleccionados como candidato.

Estos datos arrojan que SAP es mayoritario en implantación de sistemas ERP globalmente pero lo es en régimen cuasi de monopolio entre las compañías grandes lo que corrobora lo dicho anteriormente de que se ha convertido en un estándar “de facto”.

SAP ERP es propiedad de SAP SE. SAP SE es acrónimo de “Systeme, Anwendungen und Produkte in der Datenverarbeitung” (Sistemas, Aplicaciones y Productos en Procesamiento de Datos). Y SE de “Societas Europaea”. Término en el que desembocó en 2014 como sociedad europea desde el anterior SAP AG alemán.

1.4 Arquitectura de SAP ERP

SAP ERP tiene una arquitectura cliente – servidor en tres capas: [4]

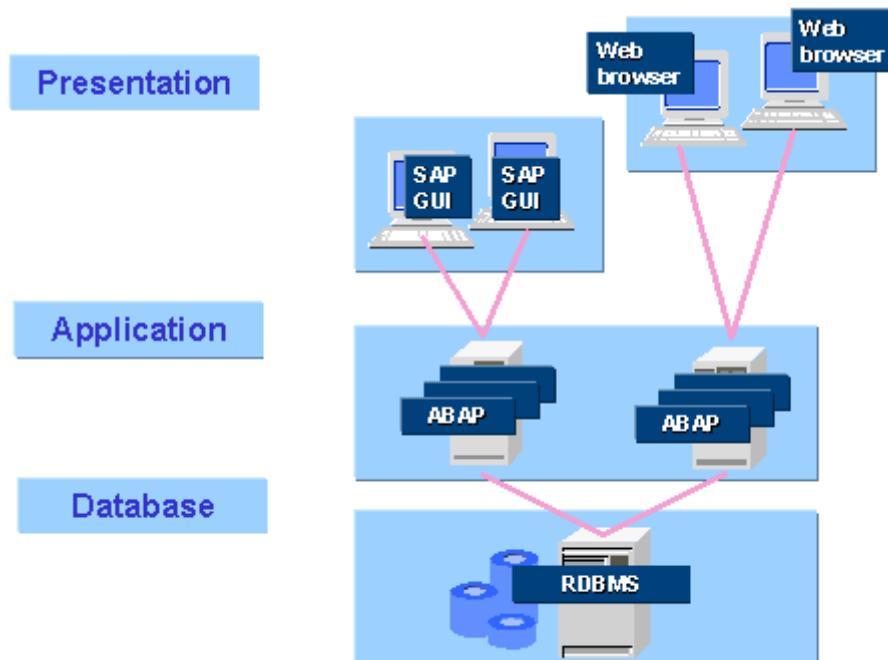


Figura 3: Arquitectura de un sistema SAP ERP.

1. Capa de base de datos. Servicios de base de datos para el salvado y recuperación de los datos empresariales. Almacena tanto datos de negocio como de configuración y programas. Es un gestor de bases de datos externo como Oracle o Microsoft SQL Server.

2. Capa de aplicación. Servicios de aplicación para el manejo de la lógica de aplicación. Corre en los servidores de aplicaciones en lenguaje de programación ABAP. En esta capa se ejecuta el SQL. ABAP tiene dos tipos de SQL:

- Open SQL: Propietario de SAP, independiente de la plataforma y con algunos restricciones.
- Native SQL: Propietario de la plataforma de la base de datos. Desaconsejado por no ser independiente de la plataforma.

3. Capa de Presentación. Servicios de presentación para la implementación del GUI.

1.5 ABAP [7]

ABAP, acrónimo de "Advanced Business Application Programming" es un lenguaje de cuarta generación, propiedad de SAP, que se utiliza para programar la mayoría de sus productos (R/3, mySAP Business suite...). Utiliza sentencias de Open SQL para conectarse con prácticamente cualquier base de datos. Cuenta con miles de funciones para el manejo de archivos, bases de datos, fechas, etc. Permite conexiones RFC (Remote Function Calls) para conectar a los sistemas SAP con cualquier otro sistema o lenguaje de programación.

ABAP fue desarrollado por SAP como lenguaje de generación de informes para SAP R/2, en los años 80, una plataforma que permitía a las grandes corporaciones construir aplicaciones de negocios para gestión de materiales y finanzas. ABAP, muy parecido al COBOL en sus orígenes, originalmente significaba "Allgemeiner Berichtsaufbereitungsprozessor", palabras alemanas para procesador genérico para la preparación de informes. En sus inicios ABAP incluía el

concepto de Bases de datos lógicas, que suministraba un alto nivel de abstracción para el acceso a bases de datos.

ABAP fue pensado como un lenguaje de programación para que los usuarios finales pudieran manipular la información, pero el 4GL se fue volviendo demasiado complicado para usuarios normales, por lo que es necesario programadores experimentados para realizar desarrollos.

ABAP se mantuvo como el lenguaje de desarrollo para la siguiente versión cliente-servidor de SAP R/3, que fue lanzada al mercado en 1992, en el que casi todo el sistema, menos las llamadas al sistema básicas estaban escritas en ABAP. En 1999, con el lanzamiento de la versión 4.6 de R/3, SAP lanzó una extensión orientada a objetos denominada ABAP Objects.

La última plataforma de desarrollo de SAP, NetWeaver, soporta ABAP y Java como lenguajes de programación.

En los años 90, se llamaba ABAP 4. Este número no era porque fuera la cuarta versión de otro, sino por ser originariamente un 4GL.

Hay una convención en cuanto a la letra de comienzo de los programas que se realizan en ABAP. Han de comenzar con la letra 'Z' o con la letra 'Y'. Todas las demás letras están reservadas por y para SAP. En particular los Yprograms en ABAP se suelen hacer para hacer test y objetos locales que nos e van a trasportar y los Zprograms para los que se van a transportar al entorno de producción. Es por ello que los programas para hacer informes en SAP, así como todos los demás que no hace SAP y se transportan empiezan por la letra 'Z' y se les conoce como programas 'Z'.

Del mismo modo, los datos en SAP nunca se modifican directamente, se hace a través de una serie de funciones que SAP previamente libera para tal fin.

SAP en particular tiene unas funciones que llama BAPI de Business Application Programming Interface que son una interfaz definida con precisión que proporciona acceso a procesos y datos en sistemas de aplicaciones empresariales como R / 3. Los BAPI se definen como métodos API de tipos de objetos de negocio de SAP. Estos tipos de objetos de negocio y sus BAPI se describen y almacenan en el Repositorio de objetos empresariales (BOR). Un BAPI se implementa como un módulo de función, que se almacena y describe en el constructor de funciones.

Los BAPI se pueden llamar dentro del sistema R / 3 desde sistemas de aplicaciones externas y otros programas. Los BAPI son el estándar de comunicación para aplicaciones empresariales.

1.6 Diccionario de SAP [4]

Se utiliza el Diccionario ABAP para crear y administrar definiciones de datos (metadatos). El diccionario ABAP permite una descripción central de todos los datos utilizados en el sistema sin redundancias. La información nueva o modificada se proporciona automáticamente para todos los componentes del sistema. Esto garantiza la integridad de los datos, la coherencia de los datos y la seguridad de los datos.

El diccionario ABAP soporta la definición de tipos definidos por el usuario (elementos de datos, estructuras y tipos de tablas). Puede crear los objetos correspondientes (tablas o vistas) en la base de datos relacional subyacente utilizando estas definiciones de datos. El diccionario ABAP describe la estructura lógica de los objetos utilizados en el desarrollo de aplicaciones y muestra cómo se asignan a la base de datos relacional subyacente en tablas o vistas.

El diccionario ABAP también proporciona funciones estándar para editar campos en la pantalla, por ejemplo para asignar ayuda de entrada a un campo de pantalla.

El diccionario de SAP al igual que todo en SAP se encuentra en la base de datos en tablas. De este modo es muy fácil acceder a él con fines de generación. Las tablas DD02L y DD03L son un buen punto de partida donde encontrar información.

1.7 Métodos modernos de control de producción [6]

El objetivo principal de estos sistemas es controlar el proceso de producción en empresas cuya actividad se desarrolla en un entorno de fabricación. La producción en este entorno supone un proceso complejo, con múltiples etapas intermedias, en las que tienen lugar procesos industriales que transforman los materiales empleados, se realizan montajes de componentes para obtener unidades de un nivel superior que a su vez pueden ser componentes de otras, hasta la terminación del producto final, listo para ser entregado a los clientes externos. La complejidad de este proceso es variable dependiendo del tipo de producto que fabriquen.

El despiece de cualquier producto complejo que se produzca es un instrumento básico de los departamentos de ingeniería de diseño para la realización de su cometido. Tanto para la especificación de las características de los elementos que componen el conjunto como para los estudios de mejora de diseños y de métodos en producción. Desde el punto de vista del control de la producción interesa la especificación detallada de los componentes que intervienen en el conjunto final, mostrando las sucesivas etapas de la fabricación o montaje del producto final, reflejando el modo en el que la misma se realiza.

Varios son los requisitos para definir esta estructura. En primer lugar cada componente o material que interviene debe de tener asignado un código que lo identifique de forma biunívoca: un único para cada elemento y a cada elemento se le asigna un código distinto. En segundo lugar, debe de realizarse un proceso de racionalización por niveles. A cada elemento le corresponde un nivel en la estructura de fabricación de un producto, asignado en sentido descendente. Así, al producto final le corresponde el nivel cero. Los componentes y materiales que intervienen en la última operación de montaje son de nivel uno, etc.

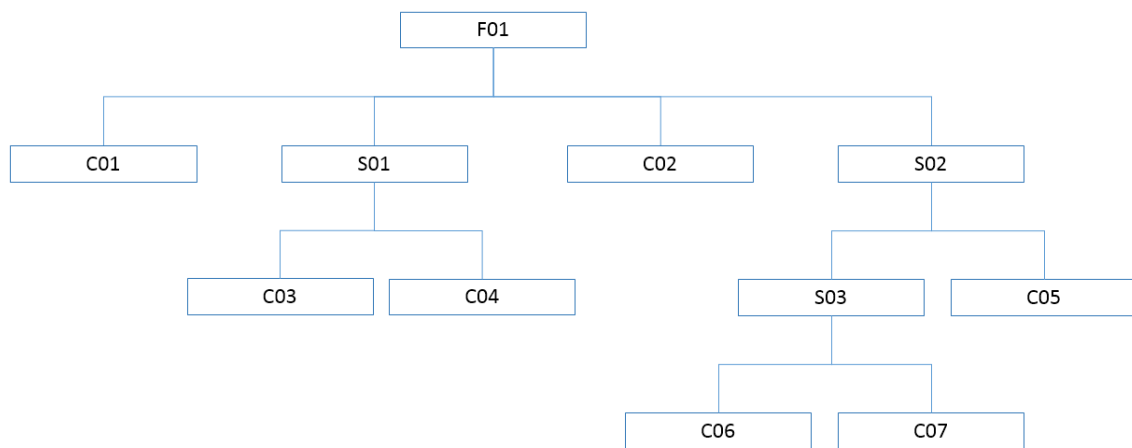


Figura 4: Arbol de producto.

En la Figura 4 podemos ver el árbol del producto. EL producto final comienza por la letra F, los productos semiterminados o de aprovisionamiento interno comienzan por la letra S y el material de compra o de aprovisionamiento externo comienzan por la letra C.

Como leímos más arriba, la representación jerárquica de una estructura multinivel está basada en la identificación biunívoca de esos elementos. Esa identificación se convierte en un código que será la base de la identificación de los materiales en el ERP. La estructura del maestro de materiales ha variado en el tiempo, del mismo modo que ha aumentado la complejidad y el ámbito de aplicación de los sistemas ERP. En los primeros sistemas ERP el maestro de materiales se componía de un solo fichero en donde se guardaba toda su información relevante. Cabe destacar que el ámbito de aplicación de estos era más limitado, siendo aplicable a una sola planta de producción. En sistemas mucho más avanzado como el sistema SAP ERP, el Maestro de materiales está repartido por múltiples tablas a medida que un mismo ERP pasa a ser compartido por todos los centros productivos y no productivos de una empresa multinacional. Encontramos tablas con información relevante a toda la empresa, otras con información relevante a los centros de producción, otras con información relevante a los almacenes, ventas, información contable, etc. El maestro de materiales de hecho replicará la estructura de la compañía.

En los métodos modernos de gestión de la producción como los programas **ERP**, para modelar el proceso de manufactura en el que se fabrica un producto acabado basado en elementos que se van procesando en pasos hasta llegar al producto final, el modelado de datos se basa fundamentalmente en tres entidades. El maestro de materiales, las listas de materiales y las hojas de ruta.

- El **maestro de materiales** contiene la identificación de todos los productos que intervienen en el proceso y ha de ser biunívoca, ya sean materiales de compra o bien materiales de fabricación interna como subconjuntos y también el producto acabado.
- Las **listas de materiales** forman una relación jerárquica en la que un producto de aprovisionamiento interno, o sea manufacturado, se compone de otros materiales que pueden ser también de aprovisionamiento interno o de aprovisionamiento externo como las compras. Esta relación jerárquica, normalmente multinivel, se explosiona para planificar el aprovisionamiento interno y externo de materiales para el proceso productivo. Hay que tener en cuenta la efectividad a la hora de explosionar. La efectividad filtra el producto de la explosión delimitando la recursividad.
- Las **hojas de ruta** son una secuencia de operaciones, ya sean manuales o de máquina que se encargan de transformar los materiales de entrada en otros. Detallan el proceso de fabricación.

2. Planteamiento del problema

2.1 Contexto del problema

En la industria manufacturera, la aparición de los ordenadores y del software ha significado un antes y un después en los procesos productivos. Esto es algo innegable pues nos ha permitido profundizar en automatización, secuenciación, mejorar la planificación de las tareas, el aprovisionamiento, etc. Los sistemas de Control de la Producción han ido evolucionando y mejorando a medida que el hardware y el software iban mejorando desde mediados del siglo pasado hasta hoy.

Durante este tiempo el software ha ido evolucionando para responder a conceptos como cumplir plazos, libre de errores, no repetir código, desarrollo rápido...

Tanto la industria manufacturera como la del software se han influido entre si aportando cada uno conceptos de la otra para mejorar ambas de forma simbiótica.

Pero hay una cosa que ha cambiado poco en este tiempo. El modelo de producción informática sigue pasando por el clásico código fuente almacenado en algún lugar, ya sea en un sistema de gestión de versiones o en la máquina del programador y la compartición no ha evolucionado mucho desde las librerías que no dejan de ser un fichero secuencial de funciones a las que se puede llamar desde el código fuente. Es por lo que, si queremos hacer código reutilizable, debemos de tenerlo en cuenta desde el principio en el diseño de los programas. Digámoslo de otra manera. No programamos pensando en la reutilización.

En la construcción de un programa, si este es grande, te puedes encontrar que es muy difícil de depurar los pasos intermedios. O dicho de otra forma, la depuración de los pasos intermedios se convierte a veces en un problema ya que normalmente cuando programamos lo hacemos pensando en el producto final y no en los pasos intermedios con lo que en el mejor de los casos nos espera una buena cantidad de horas enganchado al depurador de turno.

Por otro lado, la industria, en general, tiene una avidez de información, que sobrepasa las capacidades del sistema ERP corporativo y en la mayoría de los casos de las posibilidades del departamento de IT. Con lo que estamos asistiendo a la irrupción de un nuevo tipo de trabajo administrativo que consiste en mantener una serie más o menos estructurada de hojas de cálculo en donde se encuentran procesos que el ERP no es capaz de responder en tiempo y forma a las necesidades del usuario. Esta situación más que una ayuda, es una importante fuente de riesgo para la empresa. Estos sistemas de información no controlados, realizados por personal administrativo sin conocimientos informáticos presentan una importante cantidad de riesgos:

- Poca tolerancia al incremento del volumen de datos.
- Fallos en la lógica.
- Poco o ningún control sobre los datos de entrada.
- Riesgos al efectuar el respaldo de los datos.
- Riesgos en la seguridad.

El planteamiento del problema es intentar encontrar dentro de la industria manufacturera un modelo que ayude a la construcción de software que sea capaz de mejorar el tiempo de construcción de un proyecto software.

2.2 Objetivos concretos

El objetivo de este trabajo fin de master (TFM) es construir una solución que conste de tres elementos:

1. Una base de datos que nos permita por un lado almacenar programas, por otro los procesos y las relaciones jerárquicas entre los procesos para formar los programas.
2. Un programa que nos debe de ayudar a construir los programas de forma incremental, partiendo por la parte baja del árbol de procesos del programa e ir subiendo hasta el resultado final. Una vez concluido, nos permitirá ejecutar el programa construyendo el camino cada vez que se ejecuta a partir del árbol de procesos almacenado en la base de datos.
3. Un programa que una vez la información en la base de datos y testeada por el programa de arriba nos genere el programa objetivo independiente.

Al tener los programas estructurados de esta forma, la solución debe de ser capaz de generar código independiente de los programas individuales basados en sus árboles de procesos independientes.

Este modelo lo vamos a aplicar a un entorno SAP ERP, desarrollando mecanismos de acceso a la base de datos de SAP ERP vía RFC.

3. Estado de la cuestión

3.1 Limitaciones de ABAP y Open SQL

Un sistema de SAP ERP no solo consta del entorno productivo, normalmente hay al menos dos más. Un entorno de desarrollo y otro de control de calidad o integración. En el entorno de desarrollo se desarrollan los programas y se efectúan las parametrizaciones. Una vez hecho y probado en desarrollo que no suele tener datos 100% válidos, se transporta a un entorno de integración que suele ya tener datos 100% válidos para hacer un test completo. Tras su chequeo en éste, se transporta al entorno productivo.

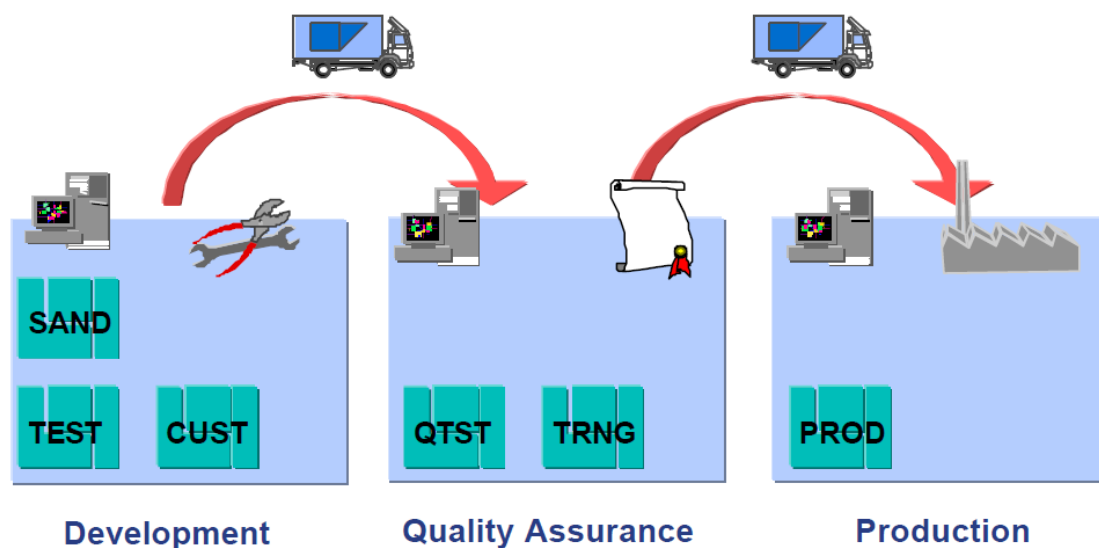


Figura 5: Sistema de transporte entre entornos SAP ERP

Este sistema es muy apropiado para hacer parametrizaciones y desarrollos complejos. Así como para tener diferenciadas las personas que actúan en diferentes entornos. Por ejemplo, se podría llegar a limitar el acceso a una subcontrata de programación externa al entorno de desarrollo impidiendo su acceso a datos confidenciales en el entorno productivo.

Pero por otro lado debido a que no se pueden hacer ciertos transportes con los usuarios conectados a el sistema productivo, los departamentos de SAP empiezan con el devenir del tiempo a programar los transportes a productivo en horas en que el usuario no esté trabajando como a altas horas de la madrugada o bien a retrasarlas en el tiempo y hacerla coincidir con algún fin de semana.

Lo que nos lleva a que para situaciones de poco riesgo y simples, este mecanismo es muy pesado y caro, como el caso de la generación de informes. Y SAP ya definió el ABAP como una herramienta de generación de informes. Curiosamente los programas ABAP comienzan con la sentencia REPORT y el nombre del programa.

Además del Open SQL disponemos, al menos en teoría, de otros dos mecanismos para el acceso a la base de datos. El Native SQL y la base de datos propiamente dicha.

Al instalar un sistema SAP ERP se hace sobre una base de datos comercial como Oracle o Microsoft SQL Server. En incluso frecuente que se haga una actualización de un sistema y se cambie la base de datos. Al instalar el motor de la base de datos, se dispone de la contraseña

de administrador del sistema y con ella se podrían acceder a los datos del sistema SAP. Esta situación no se produce en absoluto, no solo por el riesgo de producir una inconsistencia en los datos. No se accede ni siquiera en modo lectura. Esto es porque el sistema SAP no entendería los datos del motor de la base de datos pues él entiende que se realiza un acceso exclusivo, con lo que este mecanismo lo obviamos.

El Native SQL te permite acceder a características propias del motor de la base de datos sobre la que se implementa SAP ERP. No obstante como menciono anteriormente, no es tan descabellado el cambiar de motor durante las operaciones de una compañía y ello significa tener que reescribir líneas de código SQL lo que hace que tampoco se use esta opción.

Para acceder a la base de datos disponemos del Open SQL que es independiente del motor que tenga debajo, pero también tiene sus limitaciones [8]:

- Solo puedes referenciar tablas en el diccionario de SAP.
- No soporta DML como CREATE TABLE.
- No soporta características avanzadas como TRUNCATE, MERGE, ROLLUP.
- No puedes usar funciones agregadas como SUM o AVG.
- No soporta la mayoría de las funciones de columna como SUBSTR, CONCAT (||) y “expresiones case”. No lo hace en la cláusula SELECT ni en la WHERE.
- No permite cláusulas WHERE entre varias columnas así que la situación siguiente no sería posible: WHERE t1.col1 <> t1.col2
- No permite cláusulas WHERE con columnas unidas mediante LEFT y RIGHT join.
- Solo soporta dos LEFT join en una sentencia SELECT.

Como conclusión vemos que el desarrollo de informes en ABAP es un proceso lento, pesado y costoso que hace que tengamos que tratar un simple informe como si fuera un gran desarrollo y que tengamos que usar un lenguaje SQL con bastantes limitaciones con lo que se requiere una buena dosis de codificación extra sobre motores potentes como Oracle o Microsoft SQL Server.

3.2 Herramientas para hacer informes en SAP

SQVI QuickViewer

Es una herramienta gráfica que te permite definir reports de usuario.

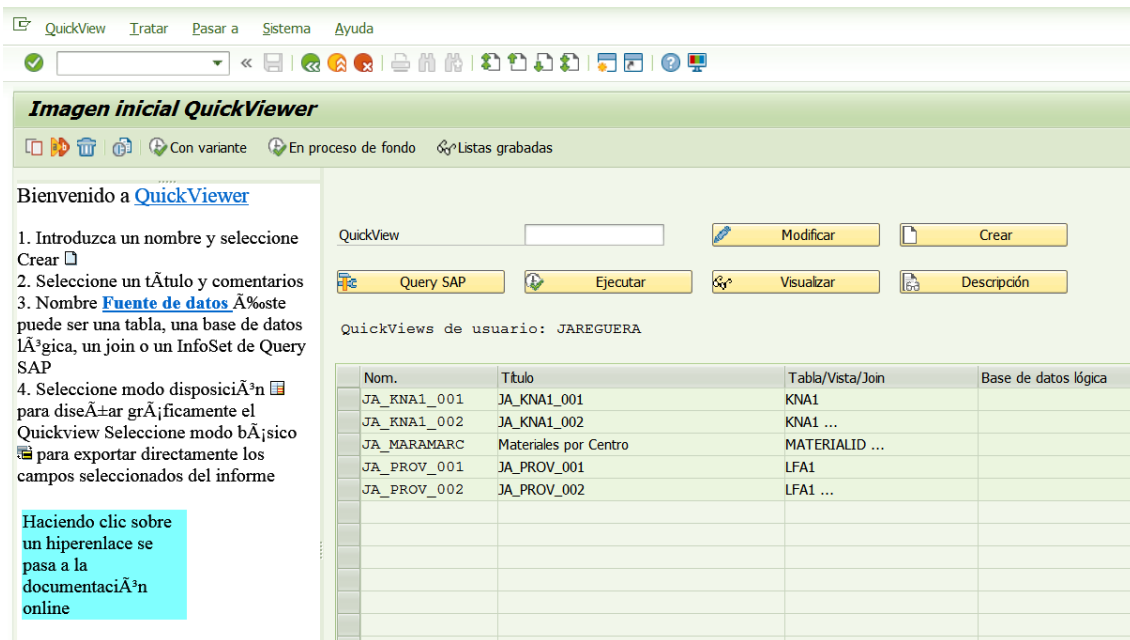


Figura 6: Transacción SQVI; imagen inicial

Es útil para definir informes sencillos de una o varias tablas de SAP.

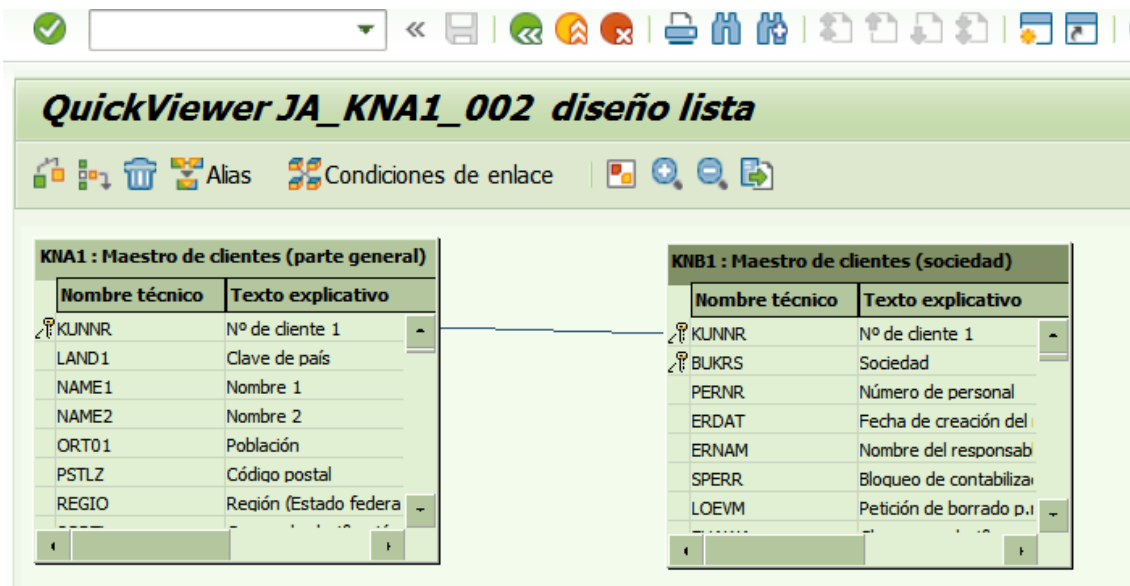



Figura 7: Transacción SQVI; Entorno gráfico de unión de tablas

Tiene las limitaciones que hemos visto sobre el Open SQL a las que hay que añadir:

- No se pueden añadir campos directamente
- Una vez hecho un informe es casi imposible redefinir la unión entre las tablas.
- No se pueden compartir fácilmente con otros usuarios.

Materiales por Centro



Materiales por Centro

Número de material	Mat	TpMt	Grupo art.	UMB	Ce.	Ce	PerfP	CaP	PIN	PzE	zHEM	LP	CAprov	ApE	zStock seguridad	UME
JA_ZPNE_001		ZPNE	2101	UN	2003			PD	P02	0	0	EX	E			0 UN
JA_ZPNV_001		ZPNV	1101	UN	2003			M0	P01	0	0	EX	E			0 UN
JA_ZPNE_002		ZPNV	2101	UN	2003			PD	P02	0	0	EX	X			0 UN
JA_ZPNC_001		ZPNC	3441	UN	2003			PD	P03	1	0	FX	F	81		0 UN
JA_ZPNC_002		ZPNC	3441	UN	2003			PD	P03	1	0	EX	F			0 UN
JA_ZPNV_002		ZPNV	1101	UN	2003			M0	P01	0	0	EX	X			0 UN
JA0033		ZPNE	01102	UN	2007			PD	ZZZ	0	0	FX	E			0 UN
JA0030		ZPNE	01102	UN	2007			PD	ZZZ	0	0	FX	E			0 UN
JA0031		ZPNE	01102	UN	2007			PD	ZZZ	0	0	FX	E			0 UN
JA0032		ZPNE	01102	UN	2007			PD	ZZZ	0	0	FX	E			0 UN
JA0034		ZPNE	01102	UN	2007			PD	ZZZ	0	0	FX	E			0 UN
JA0035		ZPNE	01102	UN	2007			PD	ZZZ	0	0	FX	E			0 UN
JA0036		ZPNE	01102	UN	2007			PD	ZZZ	0	0	FX	E			0 UN
JA0037		ZPNE	01102	UN	2007			PD	ZZZ	0	0	FX	E			0 UN
JA0038		ZPNE	01102	UN	2007			PD	ZZZ	0	0	FX	E			0 UN
JA0039		ZPNE	01102	UN	2007			PD	ZZZ	0	0	FX	E			0 UN
JA0040		ZPNE	01102	UN	2007			PD	ZZZ	0	0	FX	E			0 UN
JA0044		ZPNE	01102	UN	2007			PD	ZZZ	0	0	FX	E			0 UN
JA0041		ZPNE	01102	UN	2007			PD	ZZZ	0	0	FX	E			0 UN
JA0042		ZPNE	01102	UN	2007			PD	ZZZ	0	0	FX	E			0 UN

Figura 8: Transacción SQVI; Salida de la transacción

Esta transacción realiza las consultas sobre el entorno en la que se ejecuta y no se transportan con lo que permite realizar consultas directamente sobre el entorno productivo

ABAP/Query

Son las transacciones SQ01 a SQ03 y no vamos a profundizar sobre ello. Es un mecanismo similar al anterior pero te permite crear campos en el SELECT y que las consultas pertenezcan a un grupo de usuarios.

A cambio sin embargo te obliga a entrar en el mecanismo de los transportes y no es una herramienta muy fácil de usar.

Mi experiencia me dice que para un programador experimentado es más rápido hacer un informe completo en ABAP que en ABAP/Query.

BI Tools

Hay muchas herramientas de "Business Intelligence" en el mercado con posibilidad de extraer la información de SAP. Últimamente se las hay incluso en el modelo de SaaS (Software as a Service). En 2008, SAP adquirió Business Objects, una compañía de inteligencia de negocios y añadió sus productos a su cartera, con lo que SAP ya cuenta con su propia herramienta de BI.

Solo mencionar sobre estas herramientas que:

- No son 100% online, pues operan sobre copias de los datos que se actualizan con una cadencia dada.
- En sí, su implantación, es un proyecto importante, largo y costoso tanto en recursos como en dinero.

3.3 Funciones (RFC) que acceden a la base de datos

Para acceder a la base de datos desde fuera de SAP, SAP ERP nos da varias opciones. A mi entender la más válida para nuestro caso es la función habilitada para llamada remota o RFC Enabled.

La transacción estándar para el desarrollo de funciones en SAP es la SE37.

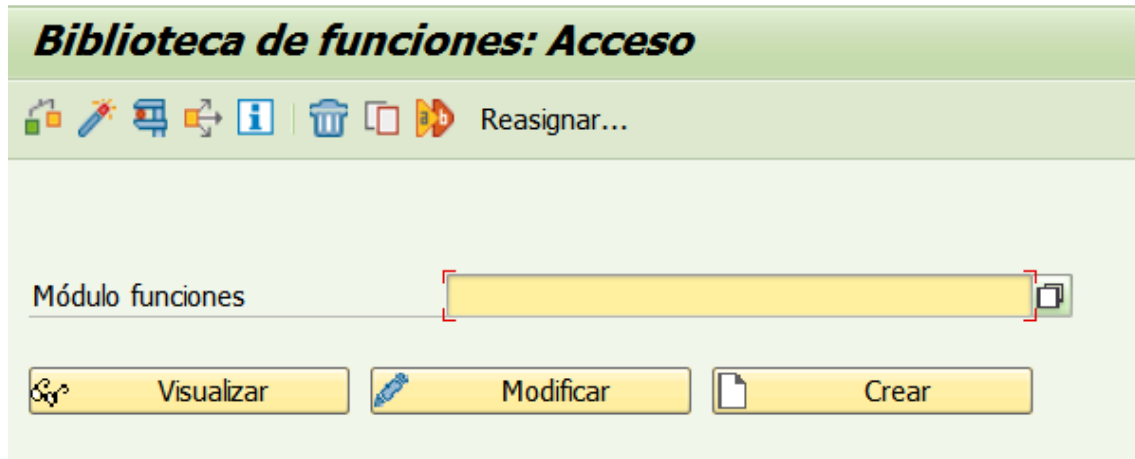


Figura 9: Imagen inicial del editor de funciones ABAP (SE37)

En la pestaña de atributos podemos ver si una función está habilitada para acceso remoto:

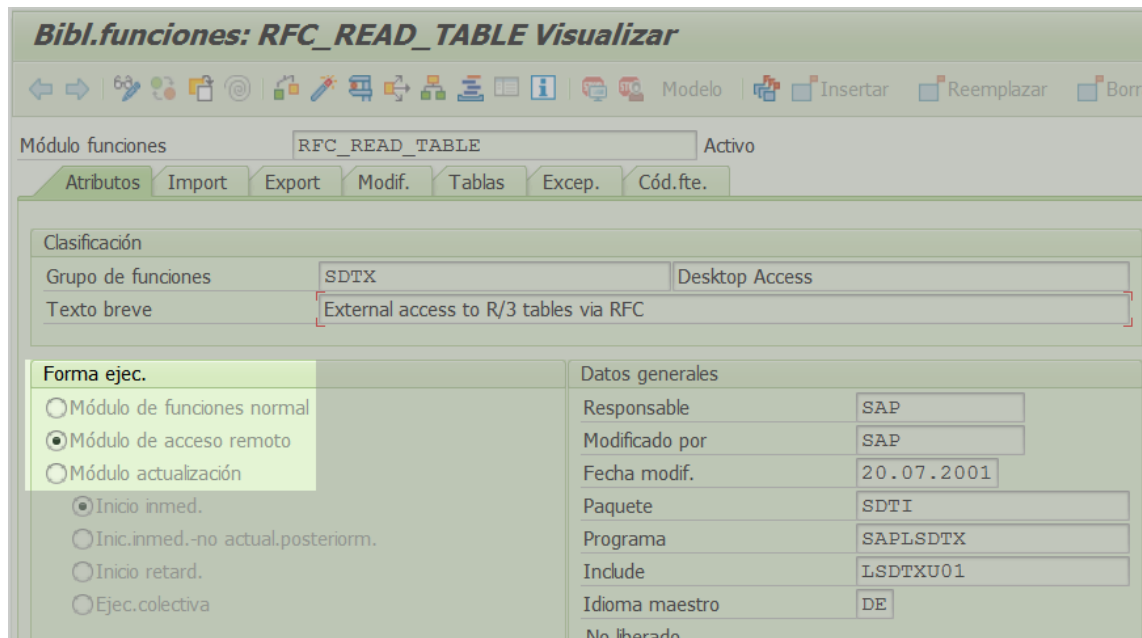


Figura 10: Pestaña Atributos (SE37)

Las funciones que están habilitadas para acceso remoto las podemos llamar con varios mecanismos como veremos más adelante y nos sirven para poder intercomunicarnos con SAP ERP desde sistemas externos a SAP.

RFC_READ_TABLE

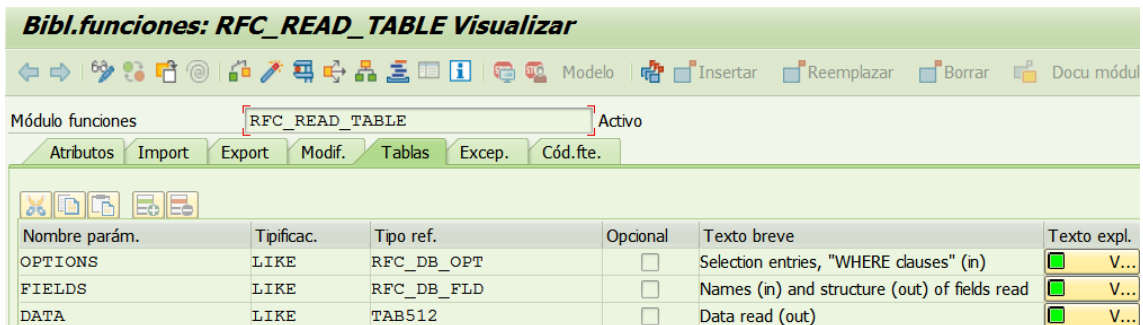
SAP ERP lleva esta función en la biblioteca estándar. O sea que se puede encontrar en todas las distribuciones de SAP. Esta función sirve para descargar datos de una sola tabla.

Admite los siguientes parámetros:



Parámetro	Descripción
QUERY_TABLE	Tabla que se desea consultar
DELIMITER	Delimitador
NO_DATA	Si 'X' solo devuelve campos y no datos

Figura 11: Parámetros import RFC_READ_TABLE



Parámetro	Descripción
OPTIONS	Cláusula WHERE
FIELDS	Cláusula SELECT; campos a devolver
DATA	Registros devueltas con Campos separados por delimitadores

Figura 12: Parámetros table RFC_READ_TABLE

Es un mecanismo simple para extraer datos de SAP ERP que funciona "out of the box" pero que tiene varios puntos flacos:

- Al no poder hacer joins, a veces las condiciones de selección son muy largas (con muchos OR) o bien se hace imposible de usar.
- El buffer de salida del campo DATA es de 512 posiciones con lo que si superamos ese límite no nos vale este procedimiento.

RFC_READ_TABLE ampliados

En principio la limitación de la función RFC_READ_TABLE que es más fácil de soslayar es el tamaño del buffer de salida. Es muy fácil encontrar en Internet funciones 'Z' que amplíen este campo a por ejemplo múltiplos de 512 posiciones y ya es mucho más difícil llegar a llenar este buffer:

Bibl.funciones: Z RFC_READ_TABLE2 Visualizar

Módulo funciones: Z RFC_READ_TABLE2 Activo

Atributos Import Export Modif. **Tablas** Excep. Cód.fte.

Nombre parám.	Tipificac.	Tipo ref.	Opcional	Texto breve
OPTIONS	LIKE	RFC_DB_OPT	<input checked="" type="checkbox"/>	RFC Table Read: Select Options / WHERE Cla...
FIELDS	LIKE	RFC_DB_FLD	<input checked="" type="checkbox"/>	RFC Table Read: Description of fields to retrie...
DATA	LIKE	ZTAB8192	<input checked="" type="checkbox"/>	Estructura para z_rfc_read_table2
			<input type="checkbox"/>	

Figura 13: Parámetros import Z RFC_READ_TABLE2

La función Z RFC_READ_TABLE2 mostrada dispone de un parámetro DATA con 8192 posiciones de longitud.

Otra solución perfectamente válida sería realizar un cambio para devolver los datos en un fichero XML con lo que no tendríamos esta restricción del tamaño en absoluto

Demás rfc enabled function modules y BAPIs

En general hay muchas más funciones que sirven para descargar datos de SAP, pero son específicas para una situación dada como por ejemplo BAPI_PRODORD_GET_LIST:

Bibl.funciones: BAPI_PRODORD_GET_LIST Visualizar

Módulo funciones: BAPI_PRODORD_GET_LIST Activo

Atributos Import Export Modif. **Tablas** Excep. Cód.fte.

Nombre parám.	Tipificac.	Tipo ref.	Opcional	Texto breve	Texto expl.
ORDER_NUMBER_RANGE	LIKE	BAPI_PP_ORDERRANGE	<input checked="" type="checkbox"/>	BAPI Interface Structure for Ranges of the Pr...	<input checked="" type="checkbox"/> V...
MATERIAL_RANGE	LIKE	BAPI_ORDER_MATERIAL...	<input checked="" type="checkbox"/>	Header Materials for Order Selection	<input checked="" type="checkbox"/> V...
PRODPLANT_RANGE	LIKE	BAPI_ORDER_PRODPLAN...	<input checked="" type="checkbox"/>	Production Plants for Order Selection	<input checked="" type="checkbox"/> V...
PLANPLANT_RANGE	LIKE	BAPI_ORDER_PLANPLAN...	<input checked="" type="checkbox"/>	Planning Plants for Order Selection	<input checked="" type="checkbox"/> V...
ORDER_TYPE_RANGE	LIKE	BAPI_ORDER_ORDER_TY...	<input checked="" type="checkbox"/>	Order Types for Order Selection	<input checked="" type="checkbox"/> V...
MRP_CNTRL_RANGE	LIKE	BAPI_ORDER_MRP_CNTR...	<input checked="" type="checkbox"/>	MRP Controllers for Order Selection	<input checked="" type="checkbox"/> V...
PROD_SCHED_RANGE	LIKE	BAPI_ORDER_PROD_SCH...	<input checked="" type="checkbox"/>	Production Schedulers for Order Selection	<input checked="" type="checkbox"/> V...
SALES_ORD_RANGE	LIKE	BAPI_ORDER_SALES_OR...	<input checked="" type="checkbox"/>	Sales Orders for Order Selection	<input checked="" type="checkbox"/> V...
SALES_ORD_ITM_RANGE	LIKE	BAPI_ORDER_SALES_OR...	<input checked="" type="checkbox"/>	Sales Order Items for Order Selection	<input checked="" type="checkbox"/> V...
WBS_ELEMENT_RANGE	LIKE	BAPI_ORDER_WBS_ELEM...	<input checked="" type="checkbox"/>	WBS Elements for Order Selection	<input checked="" type="checkbox"/> V...
SEQ_NO_RANGE	LIKE	BAPI_ORDER_SEQ_NO_R...	<input checked="" type="checkbox"/>	Sequence Numbers for Order Selection	<input checked="" type="checkbox"/> V...
ORDER_PRIO_RANGE	LIKE	BAPI_ORDER_PRIO_RAN...	<input checked="" type="checkbox"/>	Order Priority for Order Selection	<input checked="" type="checkbox"/> V...
ORDER_HEADER	LIKE	BAPI_ORDER_HEADER1	<input checked="" type="checkbox"/>	Order header data	<input checked="" type="checkbox"/> V...

Figura 14: Parámetros table BAPI_PRODORD_GET_LIST

3.4 Middleware sobre el que ejecutar RFC

Obviamente necesitamos un mecanismo de transporte para comunicar con SAP. En los últimos tiempos está cada vez más implantado el mecanismo WEB SERVICES pero anteriormente, el primero que se pudo usar fue un API del lenguaje de programación C. Además es muy fácil de usar el OLE (Object Linking and Embedding) de Microsoft y también hay un API de Java.

MS OLE

Normalmente se instala conjuntamente con el SAP GUI a no ser que se indique lo contrario. Hay que usar un objeto *SAP.Functions* que se encargará de ejecutar la función sobre el servidor de SAP

```
Set SAPFunction = CreateObject("SAP.Functions")
SAPFunction.logfilename = SClogfilename
SAPFunction.loglevel = SCloglevel
Set SAPConn = SAPFunction.Connection
SAPConn.tracelevel = SCTracelevel
SAPConn.ApplicationServer = SCAApplicationServer 'Requerida
SAPConn.SystemNumber = SCSystemNumber 'Requerida
SAPConn.System = SCSystem 'Requerida
SAPConn.Client = SCClient 'Requerida
SAPConn.User = SCUser 'Requerida
SAPConn.Password = SCPassword 'Requerida
SAPConn.language = SCLang 'Requerida
If Not SAPConn.logon(0, True) Then
    Conecta = False
    Exit Function
Else
    Conecta = True
End If
```

Figura 15: Script VBA para conectar a SAP via SAP.Functions OLE Object

Java

SAP dispone de un software que te puedes descargar de SAP con licencia e instalar localmente que es el Java Connector que actualmente está en la versión 3.

El SAP Java Connector (SAP JCo) es un componente de middleware que permite el desarrollo de componentes y aplicaciones compatibles con SAP en Java. SAP JCo admite la comunicación con el servidor SAP en ambas direcciones: llamadas entrantes (llamadas Java ABAP) y llamadas salientes (llamadas ABAP Java).

SAP JCo se puede implementar con aplicaciones de escritorio y con web services.

```
require 'C:/sapjco/sapjco3.jar'
java_import com.sap.conn.jco.JCoTable
java_import com.sap.conn.jco.JCoDestinationManager
java_import com.sap.conn.jco.JCoContext
```

Figura 16: Script Ruby para declarar clases para Java Connector 3.0

Web Services

SAP tiene la transacción SOAMANAGER para manejar los WEB Services que se realizan para un grupo de funciones determinado.

Es una funcionalidad que no necesita la instalación de software adicional en el servidor de SAP, funciona correctamente “out of the box”.

4. Resolución

Se ha construido un sistema inspirado en un sistema básico de control de la producción. No es el objetivo de este trabajo fin de master describir la estructura de un sistema ERP ni su evolución en el tiempo sino la parte de este que puede ser útil al objetivo de este trabajo que es la construcción y generación de código basado en estructuras multinivel inspiradas en dicho concepto.

Vamos a descomponer un programa en un árbol de procesos elementales. Cada uno de estos procesos elementales va a tomar el lugar de un material y su relación que al final nos va a dar la secuencia de ejecución la vamos a informar como una estructura de producto o de fabricación, en inglés BOM.

Se ha diseñado una solución basada en base de datos que sirve para generar programas con árboles de procesos que pueden ser compartidos por varios programas y que nos ayude a:

- Agilizar la producción de software
- Servir como base de conocimiento
- Fomentar la reutilización de código
- Facilitar la estructuración y documentación del software

4.1 Lenguaje, base de datos y middleware

JRuby [9][10][11][12][13][14][15]

Para el desarrollo de la solución se ha optado por JRuby pues siendo Ruby el lenguaje más usado en el Master, el Jruby además mejora en su conexión con SAP por facilitar el uso de las librerías Java JCo3 de SAP manteniendo sus demás ventajas como las plantillas eRB.

SQLite3 [16]

SQLite3 ha sido la opción escogida. Al ser una opción libre, que permite bases de datos sin servidor y por permitir tener campos en tablas débilmente tipados.

SQLite es una librería que implementa un motor de base de datos SQL autónomo, sin servidor, sin configuración y transaccional. El código para SQLite está en el dominio público y por lo tanto es gratuito para su uso para cualquier propósito, comercial o privado. SQLite es la base de datos más ampliamente implementada en el mundo con más aplicaciones de las que podemos contar, incluyendo varios proyectos de alto perfil.

SEQUEL [17]

Al estar Jruby basado en Java y no en C, las librerías de manejo de la base de datos son jdbc de Java. El uso de esta librería no se puede considerar muy “Ruby” con lo que se ha usado SEQUEL que es un ORM que simplifica muchísimo el manejo de la base de datos.

Sus características son:

- Sequel proporciona seguridad de subprocesos (hilos), pooling de conexiones y un DSL conciso para la construcción de consultas SQL y esquemas de tablas.
- Sequel incluye una capa de ORM completa para cartografiar registros a objetos Ruby y gestionar registros asociados.
- Sequel es compatible con funciones de base de datos avanzadas como sentencias preparadas, variables host, procedimientos almacenados, puntos de guardado, confirmación de dos fases, aislamiento de transacciones, configuraciones maestro / esclavo y fragmentación de bases de datos.

- Actualmente, Sequel tiene librerías para ADO, Amalgalite, CUBRID, DataObjects, IBM_DB, JDBC, MySQL, Mysql2, ODBC, Oracle, PostgreSQL, SQLAnywhere, SQLite3, Swift y TinyTDS.

SAP JCo3 [18]

SAP Java Connector (SAP JCo) es un componente de middleware que permite el desarrollo de componentes y aplicaciones compatibles con SAP en Java. SAP JCo admite la comunicación con el servidor SAP en ambas direcciones: llamadas entrantes (llamadas Java ABAP) y llamadas salientes (llamadas ABAP Java).

SAP JCo se puede implementar con aplicaciones de escritorio y con aplicaciones de servidor web.

Actualmente SAP distribuye la versión 3. Esta versión no es compatible con las versiones anteriores.

4.2 Descripción del Programa a generar

El programa objetivo a generar es una secuencia de procesos elementales independientes que operan sobre una base de datos SQLite que se genera al empezar a ejecutarse que se llamará *WorkDB* de ahora en adelante. Esa base de datos es como la “memoria” del programa ya que no hay, a excepción del handle de la base de datos, variables de instancia. Los procesos se ejecutan secuencialmente compartiendo esa base de datos y al acabar se puede exportar el resultado o efectuar alguna acción. Idealmente todos los procesos producen una tabla y se alimentan de las tablas que generan los procesos que están en su estructura. De hecho los procesos tienen un campo de “tabla creada” que normalmente se pasa como parámetro a la plantilla del proceso. Claro está, en el caso del código puro, esto es más difícil de mantener lo que requiere especial cuidado al hacer estos procesos elementales de código puro. Pero a su vez, esto da mucha potencia al modelo con lo que no se puede evitar.

4.3 Descripción de la solución

Vamos a hacer un generador simple como se describe en el capítulo 4 del libro de la asignatura “Generación automática de código”[9] pero que a partir de plantillas ERB va a generar un código JRuby como se muestra en el capítulo 5 de dicho libro. La entrada de ese generador con el código no va a ser un fichero fuente sino una base de datos y la lógica será calculada con la estructura presente en las tablas de dicha base de datos.

La solución consta de una base de datos y de dos programas. La base de datos va a contener la información sobre los programas sus procesos constituyentes, la relación entre ellos y su efectividad. El programa **Desarrollador** que nos va a ayudar a construir el programa objetivo a partir de los procesos elementales jerarquizados que hemos incluido en la base de datos. Nos va a permitir su construcción y depuración incremental, construyendo el camino cada vez que se va a ejecutar desde el principio y permitiéndonos ir probando de uno en uno los procesos elementales y examinando los resultados intermedios. Una vez depurado el programa, este mismo programa **Desarrollador** nos va a permitir ejecutarlo o bien podemos usar el programa **Generador** para construir el producto final, o sea el programa generado, con la información guardada en la base de datos y nos permitirá su ejecución independientemente de la base de datos de la solución.

Como en los métodos modernos de gestión de producción, los datos maestros se guardan en una base de datos. Esta base de datos guardará información sobre cada proceso elemental y sobre las relaciones de procesos. El maestro de materiales lo vamos a asimilar a los procesos

elementales, la estructura de producto guarda las relaciones padre hijo que van a dar la secuencia final de ejecución de los procesos. La efectividad va a filtrar qué procesos se ejecutan y qué procesos no se ejecutan para una misma lista, con lo que tenemos listas de procesos configurables, o lo que es lo mismo un proceso, dependiendo de en qué programa se ejecuta va a poder tener una serie de procesos hijos u otra más o menos diferente.

La base de datos con todos los datos maestros de los procesos elementales y su relación que nos va a permitir generar el programa y que es accesible solo al **Desarrollador** y al **Generador** se va a llamar de ahora en adelante *GenDB*.

El proceso consiste en la ejecución de un proceso elemental presente en la base de datos para ello se realizan los siguientes pasos:

- Explosión del proceso elemental para la obtención de los procesos elementales de que consta a partir de la información alojada en *GenDB*.
- Obtención de la secuencia de ejecución a partir de la explosión.
- Creación de la base de datos del proceso *WorkDB*.
- Ejecución de la secuencia de procesos obtenida más arriba desde la información en la base de datos.

En resumen, el programa **Desarrollador** es una utilidad que nos sirve para depurar la estructura del código a generar que hemos informado en la base de datos *GenDB*. Y el programa **Generador** nos sirve para obtener la solución esperada a partir de dicha base de datos y de las plantillas ERB de los tipos de proceso desarrollados.

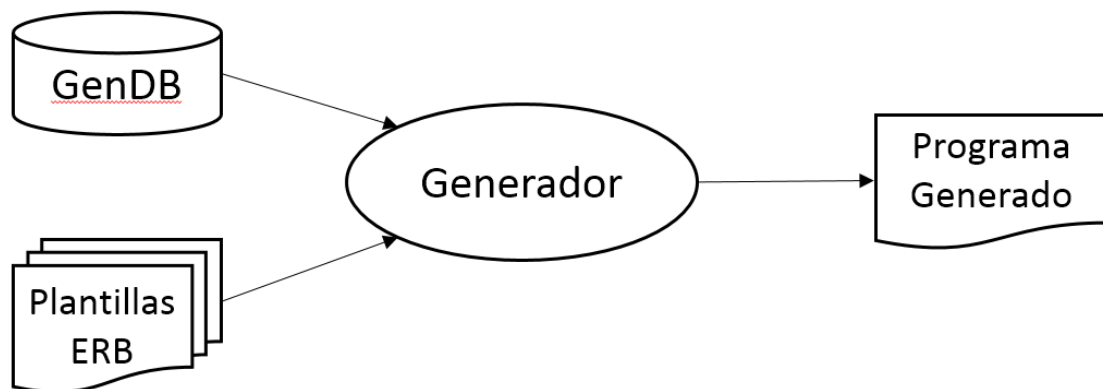


Figura 17: Diagrama del proceso de generación.

4.4 Base de datos GenDB

La entrada de lógica en nuestro proceso de generación no es un fichero plano. Es una base de datos. Se le llama *GenData.sqlite* y reside en el directorio donde se ejecuta tanto el programa **Desarrollador** como el programa **Generador**.

La base de datos consta de tres tablas relacionadas según la figura 18.

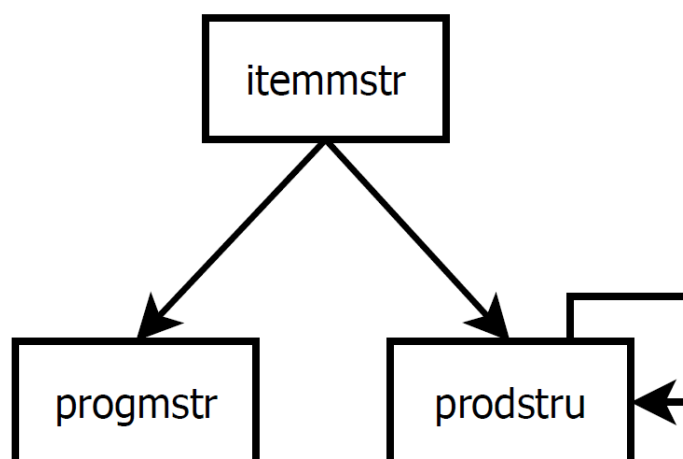


Figura 18: Relaciones de las tablas de la base de datos GenData.sqlite3

Usamos la flecha para indicar una relación 1 a muchos.

progmr: Contiene el nombre del programa y el proceso de comienzo para iniciar la explosión del árbol de procesos. Entre la información relevante del programa está su descripción y el directorio por defecto de la aplicación generada. Este directorio es útil solo para el programa desarrollador. Es donde se van colocando los ficheros relevantes al programa.

<i>Campo</i>	<i>Tipo</i>	<i>Descripción</i>
progmr		Maestro de programas
prog_no	TEXT	Nombre de programa
prog_desc	TEXT	Descripción del programa
item_no	TEXT	Proceso inicial
prog_path	TEXT	Directorio por defecto

Tabla 19: Esquema de la tabla progmr

itemmstr: Contiene la información relevante al proceso elemental. El tipo de proceso le indica la forma en que se ha de tratar el proceso. Si el proceso genera una tabla, debe de ir indicada aquí, como el caso de las tablas que se generan desde una sentencia SELECT o desde una importación de datos. El campo de código se usa para almacenar o bien el path a un fichero con código JRuby o una sentencia SQL o info relativa a donde se encuentra el fichero de donde se va a efectuar la importación de datos.

<i>Campo</i>	<i>Tipo</i>	<i>Descripción</i>
itemmstr		Maestro de Procesos elementales
item_no	TEXT	Nombre Proceso
item_desc	TEXT	Descripción
item_type	TEXT	Tipo de Proceso
item_table	TEXT	Tabla que produce en su caso
item_code	TEXT	El código que se ejecuta.

Tabla 20: Esquema de la tabla itemmstr

prodstru: Contiene las relaciones padre – hijo entre los procesos. A la hora de la explosión hay que tener en cuenta que no todos los procesos hijo son efectivos. Hay que tener en cuenta la efectividad. Si la efectividad *prog_efec* está informada indicará que esa relación es efectiva solo para ese programa. Si no está informada será efectiva para todos los programas que no excepto los que sí poseen una entrada con su nombre.

Al tener en cuenta la efectividad a la hora de la explosión obtenemos explosiones diferentes según el programa que las ejecute con lo que el camino cambia. Un proceso ahora puede tener varias fuentes de datos diferentes con lo que las posibilidades de reutilización aumentan considerablemente.

<i>Campo</i>	<i>Tipo</i>	<i>Descripción</i>
prodstru		Rlaciones entre los Procesos elementales
item_no	TEXT	Nombre del Proceso Padre <-itemmstr
item_no_comp	TEXT	Nombre del Proceso Hijo <- itemmstr
prog_efec	TEXT	Efectividad <- progmsr

Tabla 21: Esquema de la tabla *prodstru*

4.5 Tipos de proceso

Se han desarrollado unos tipos de proceso que se pueden interconectar para producir e programa. Normalmente pueden ser de entrada, internos o de salida.

Los procesos de entrada responden a una plantilla que importa datos a una tabla de la base de datos *WorkDB*. Pueden importar desde un fichero plano, una hoja de Microsoft Excel o desde SAP mediante una sentencia Open SQL de ABAP.

Los internos normalmente son sentencias SELECT de SQLite.

Los externos exportan una tabla que se ha obtenido previamente en la base de datos *WorkDB*.

Los procesos ruby son procesos que no tienen limitaciones pudiendo ser de entrada, salida o internos. Sin embargo tienen una limitación. La única variable que pueden obtener del programa es la base de datos *WorkDB*. Todos los datos que manejen han de estar sobre la base de datos *WorkDB*.

Tipo	Descripción	Lógica	Parámetro
fictabin	Importar Fichero plano separado por Tabulaciones Con Cabeceras	Importar fichero con nombre. Comprobar que el formato es correcto. Crear y popular la tabla.	Path al fichero
fictabout	Exportar Fichero plano separado por Tabulaciones Con Cabeceras	Exportar fichero con nombre.	Path al fichero
sapsql	Importar Query desde SAP	Se ejecuta y crea la tabla.	Open SQL Select
ruby	Código Ruby	Se ejecuta.	Código Ruby
sql	Sentencia Select	Se ejecuta y crea la tabla.	SQLite Select
xlolein	Excel importar tabla	Importa hoja excel con formato de tabla con columnas y con los nombres de campo en la primera fila.	Path al libro excel y hoja

Tabla 22: Tabla con los tipos de proceso.

fictabin: Importa una tabla desde un fichero plano con las columnas separadas por tabulaciones. La primera línea del fichero contiene los nombres de los campos. Al ser una plantilla solo hay que actualizar el nombre de la tabla generada y el path donde se aloja el fichero plano a importar.

fictabout: Exporta una tabla desde *WorkDB* a un fichero plano. Es una plantilla con lo que generará un fichero plano con el mismo nombre de la tabla en el path que tiene también informado.

ruby: Es un código libre de Jruby con la limitación de que solo hay una variable de instancia que recibe del programa principal que es la base de datos *WorkDB* y que solo puede actuar, de hecho, sobre la base de datos *WorkDB*. Esto actúa como un comodín.

sapsql: Importa una tabla en *WorkDB* desde una sentencia OpenSQL que le pasamos a SAP por RFC a la función *Z_RFC_SQL*. Los parámetros son el nombre de la tabla a crear y la sentencia Open SPL.

sql: Es un proceso de creación de tabla desde otras tablas en la base de datos *WorkDB*. Debe de ser el principal actor en esta representación. Tiene como parámetros la sentencia Select y la tabla a crear. Realmente lo que hace es construir una sentencia “insert into *tabla* from *sentencia select*”. El haberlo hecho de esta forma es por paralelismo con los demás tipos que reciben los parámetros de la forma tabla y código o path.

xltabin: Importa una tabla desde un libro de Microsoft Excel. Sus parámetros son el path al fichero y el nombre de la hoja. Se importa con el mismo nombre de la hoja.

4.6 Proceso sapsql

Merece especial mención el proceso elemental tipo sapsql. Este proceso usa RFC a través de JCo3 para comunicarse con SAP ERP.

Para poder usar parámetros variables en la sentencia SELECT, hemos desarrollado un procedimiento inspirado en la forma en que ABAP solucionó este problema. ABAP usa en la sentencia SELECT una extensión de la de la cláusula WHERE que es la FOR ALL ENTRIES IN NOMBRETABLA. Esta extensión permite a SAP generar sentencias SELECT que manda al servidor con una serie de condiciones unidad mediante el operador OR, una por cada registro de la tabla NOMBRETABLA suministrada.

Así que el proceso sapsql hace lo mismo. Idéntica la extensión de la cláusula WHERE, la elimina al pasarla a la RFC. Expande la sentencia WHERE de forma que la repite tantas veces como registros haya en la tabla envuelta en paréntesis y unidas mediante un operador OR.

Para obtener la información de la base de datos de SAP se ha desarrollado la función de acceso remoto *Z_RFC_SQL* que se mencionará más adelante.

4.7 Programa Desarrollador

Está concebido para servir de ayuda al desarrollo y la integración de los procesos elementales. Este programa tiene acceso a la base de datos *GenDB* y al recibir como argumento el nombre de un programa es capaz de obtener una secuencia de ejecución de procesos elementales.

El programa desarrollador también tiene, al igual que el programa Generado, una conexión a *WorkDB*. Una vez obtenida la secuencia de ejecución, la almacena en la base de datos *WorkDB* con un flag que indica si el proceso se ha ejecutado. De esta forma se pueden ir construyendo

los arboles de procesos de forma incremental, pues al irse ejecutando desde el principio, se van generando resultados intermedios en *WorkDB* y al lanzarlo de nuevo empieza por el primer proceso no ejecutado obviando la ejecución de los procesos desde el principio. Esto nos sirve para el desarrollo incremental de los procesos, ya que no repites en el desarrollo etapas que ya antes has ejecutado.

Para volver a ejecutar los procesos desde el principio, se hace necesario borrar la base de datos *WorkDB* del directorio de la aplicación.

Desarrollador
- @conn_gen: Sequel::Database
- @conn_work: Sequel::Database
+ ejec_programa(programa:String)
- calc_sec(proceso:String)
- crea_tabla(item_table:String, campos:Array, filas:Array)
- func_rec(mi_proceso:String, nivel:Integer)
- proc_fictabin(item_no:String, item_code:String, item_table:String)
- proc_fictabout(item_no:String, item_code:String, item_table:String)
- proc_ruby(item_no:String, item_code:String, item_table:String)
- proc_sapsql(item_no:String, item_code:String, item_table:String)
- proc_sql(item_no:String, item_code:String, item_table:String)
- proc_xltabin(item_no:String, item_code:String, item_table:String)

Figura 23: Clase Desarrollador.

- @conn_gen: Sequel::Database

Variable de instancia privada con la conexión a la base de datos de la solución donde se almacenan la información de programas, procesos y relaciones.

- @conn_work: Sequel::Database

Variable de instancia privada con la base de datos de los datos intermedios.

+ ejec_programa(programa:String)

Método público que comienza el proceso de ejecución incremental de un programa.

- *programa*: cadena con el programa a ejecutar.

- calc_sec(proceso:String)

Método privado que calcula la secuencia de ejecución de los procesos elementales basados en una llamada a la función recursiva *func_rec*.

- *proceso*: cadena con el proceso raíz por dónde empezar la recursividad.

- crea_tabla(item_table:String, campos:Array, filas:Array)

Método privado que crea una tabla en *WorkDB* de datos intermedios.

- *item_table*: cadena con el nombre de la tabla a crear.
- *campos*: array con los nombre de los campos de la tabla a crear.
- *filas*: array con el contenido de las filas a insertar en la tabla a crear en la base de datos *WorkDB*.

- *func_rec(mi_proceso:String, nivel:Integer)*

Método privado recursivo que nos sirve para calcular la secuencia de ejecución de los procesos elementales.

- *poceso*: cadena con el nombre de proceso que se usa en este nivel de recursividad.
- *nivel*: entero conteniendo el número de la profundidad de la recursividad.

- *proc_fictabin(item_no:String, item_code:String, item_table:String)*

Método privado que importa un fichero plano separado por tabulaciones con los nombres de los campos en el primer registro a una tabla en *WorkDB*.

- *item_no*: nombre del proceso.
- *item_code*: ubicación del fichero plano.
- *item_table*: nombre de la tabla que se creará en *WorkDB*.

- *proc_fictabout(item_no:String, item_code:String, item_table:String)*

Método privado que exporta un fichero plano separado por tabulaciones con los nombres de los campos en el primer registro desde una tabla en *WorkDB*.

- *item_no*: nombre del proceso.
- *item_code*: ubicación del fichero plano.
- *item_table*: nombre de la tabla que se exportará desde *WorkDB*.

- *proc_ruby(item_no:String, item_code:String, item_table:String)*

Método privado que ejecuta un fichero con código jruby hecho para ser ejecutado con esta solución. Se entiende que usará la variable de instancia *@conn_work*.

- *item_no*: nombre del proceso.
- *item_code*: ubicación del fichero plano conteniendo el código jruby.
- *item_table*: nombre de la tabla que se creará en *WorkDB* en su caso.

- *proc_sapsql(item_no:String, item_code:String, item_table:String)*

Método privado que ejecuta la sentencia OpenSQL que hay en *item_code*, llama a la función *Z_RFC_SQL* y deposita los datos XML obtenidos en la tabla *item_table* de *WorkDB*.

- *item_no*: nombre del proceso.
- *item_code*: código OpenSQL SELECT que se ejecutará contra el servidor de SAP ERP.
- *item_table*: nombre de la tabla que se creará en *WorkDB*.

- *proc_sql(item_no:String, item_code:String, item_table:String)*

Método privado que ejecuta la sentencia SELECT de SQLite que hay en *item_code* sobre la base de datos *WorkDB* guardando el resultado en la tabla creada con nombre en *item_table*.

- *item_no*: nombre del proceso.
- *item_code*: sentencia SELECT SQLite3 que se ejecutará en la base de datos *WorkDB*.
- *item_table*: nombre de la tabla que se creará en *WorkDB*.

- *proc_xlabin(item_no:String, item_code:String, item_table:String)*

Método privado que importa una tabla en *WorkDB* de nombre *item_table* desde una hoja Excel ubicada en *item_code*, desde la hoja con nombre en *item_table* y con los nombre de campo situados en la primera columna de dicha hoja de cálculo.

- *item_no*: nombre del proceso.

- *item_code*: ubicación del libro de Microsoft Excel.
- *item_table*: nombre de la tabla que se creará en *WorkDB*.

4.8 Programa Generador

La generación consiste en obtener un proceso como se ha indicado arriba, como una secuencia de ejecución de procesos elementales. Ejecutando los siguientes pasos:

1. Generación de la cabecera del programa generado con la conexión a la base de datos *WorkDB*.
2. Obtención de la secuencia de ejecución de procesos elementales con los datos de la base de datos *GenDB*.
3. Generación del proceso principal y de los procesos elementales con la información de *GenDB* y plantillas eRB por tipo de proceso.

Generador
- @conn_gen: Sequel::Database
+ genera(programa:String)
- func_rec(proceso:String, resultado:Array, nivel:Integer)
- proceso_funciones(miarr:Array)

Figura 24: Clase Generador.

- @conn_gen: Sequel::Database

Variable de instancia privada con la conexión a la base de datos de la solución donde se almacenan la información de programas, procesos y relaciones. *GenDB*.

+ genera(programa:String)

Método público que efectúa la generación del programa en base a la información contenida en la base de datos *GenDB*.

- *programa*: cadena conteniendo el nombre del programa a generar.

- func_rec(proceso:String, resultado:Array, nivel:Integer)

Método privado recursivo que nos sirve para calcular la secuencia de ejecución de los procesos elementales.

- *proceso*: cadena con el nombre de proceso que se usa en este nivel de recursividad.
- *resultado*: array en donde se almacena la secuencia de ejecución.
- *nivel*: entero conteniendo el número de la profundidad de la recursividad.

- proceso_funciones(miarr:Array)

Método privado que genera el código de los métodos que se ejecutan en el programa mediante plantillas eRB.

- *miarr*: Array con la secuencia de procesos elementales que se van a generar.

4.7 Programa Generado

Generada
- @conn_work: Sequel::Database
+ proceso
- crea_tabla(item_table:String, campos:Array, filas:Array)
- una clase con el nombre de cada proceso...

Figura 25: Clase Generada.

Obviamente este programa coincide con el **Desarrollador** solo que no tiene referencia a *GenDB* y que los métodos están secuenciados en proceso y se encuentran generados, privados, tras el método privado *crea_tabla*

4.10 Z_RFC_SQL

Como vimos arriba las RFC proveídas por SAP dan un abanico muy pobre de posibilidades para acceder a la base de datos de SAP ERP. Es por ello que basado en el programa ZSQL que mencionamos antes se ha desarrollado un módulo de funciones con acceso remoto llamado Z_RFC_SQL. Este módulo de funciones es capaz de aceptar una sentencia SELECT en open SQL y devuelve un fichero XML con los datos obtenidos.

5 Casos de Estudio

Vamos a hacer dos. Uno primero sin acceso a SAP para centrarnos en las jerarquías de procesos elementales y otro con acceso al sistema SAP. Se describirán todos los programas, todos los procesos elementales que los constituyen las relaciones entre ellos que al final nos dará la secuencia de ejecución

5.1 Caso de estudio sin SAP

Vamos a empezar por un caso de estudio simple sin acceso a SAP. Solo para explorar la jerarquía de procesos.

Vamos a hacer unos programas que a partir de dos tablas, una con números y sus numerales y otra con números y sus cardinales podamos obtener otra tabla con los números, sus cardinales y sus numerales. Para introducir variación y poder reutilizar vamos a usar entradas por fichero plano o por hojas de cálculo y vamos a incluir un proceso que modifica las tablas de entrada para obtener un resultado modificado. En este caso será poner el primer carácter del cardinal en mayúscula.

Programas a realizar:

- numeros_00: con entrada desde fichero plano y minúsculas
- numeros_01: con entrada desde fichero plano y conversión
- numeros_02: con entrada desde hoja de cálculo y minúsculas
- numeros_04: con entrada desde hoja de cálculo y conversión

Procesos elementales:

Los procesos elementales para fomentar la claridad están precedidos en el nombre por el tipo de proceso elemental.

- fictabin_cardinales: Entrada a tabla en WorkDB desde fichero plano separado por tabulaciones de la tabla cardinales.
- fictabin_ordinales: Entrada a tabla en WorkDB desde fichero plano separado por tabulaciones de la tabla ordinales.
- xltabin_cardinales: Entrada a tabla en WorkDb desde hoja de cálculo de la tabla cardinales.
- xltabin_ordinales: Entrada a tabla en WorkDb desde hoja de cálculo de la tabla ordinales.
- ruby_cardinales_m: programa jruby que actualiza el compo cardinal de la tabla cardinales convirtiendo el primer carácter a mayúsculas.
- sql_numeros: Sentencia SELECT que genera la tabla resultado.
- fictabout_numero: Salida a fichero plano separado por tabulaciones de la tabla numeros.

Estructura de los procesos elementales:

Las estructuras jerárquicas de los programas con los procesos elementales pasamos a verlas de programa en programa a continuación.

numeros_00

Procesos:

1. `fictabin_cardinales`: importa a WorkDB fichero plano con dos columnas, número y su cardinal.
2. `fictabin_ordinales`: importa a WorkDB fichero plano con dos columnas, número y su ordinal
3. `sql_números`: proceso SQL que obtiene una tabla con números, su cardinal y su ordinal.
4. `fictabout_numero`: exporta la tabla obtenida en el proceso anterior a fichero plano

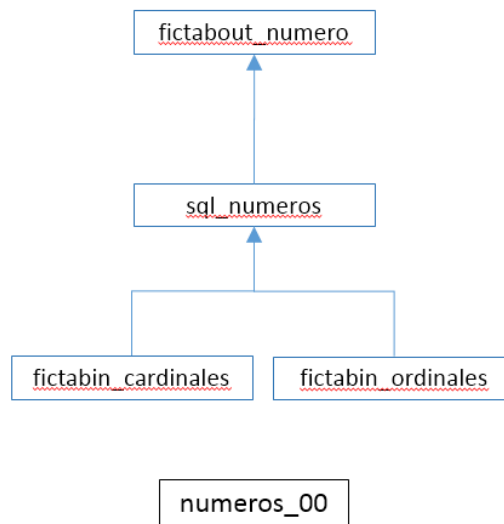


Figura 26: Estructura de procesos elementales de `numeros_00`

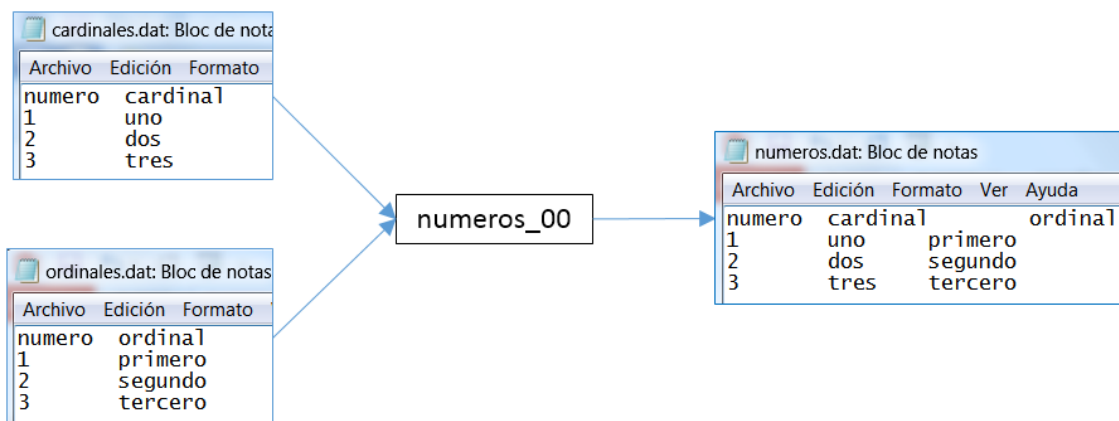


Figura 27: Descripción funcional de `numeros_00`

numeros_01

Procesos:

1. fictabin_cardinales: importa a WorkDB fichero plano con dos columnas, número y su cardinal.
2. fictabin_ordinales: importa a WorkDB fichero plano con dos columnas, número y su ordinal
3. ruby_cardinales_m: proceso ruby que convierte el primer carácter del campo cardinal de la tabla cardinales a mayúscula
4. sql_números: proceso SQL que obtiene una tabla con números, su cardinal y su ordinal.
5. fictabout_numero: exporta la tabla obtenida en el proceso anterior a fichero plano

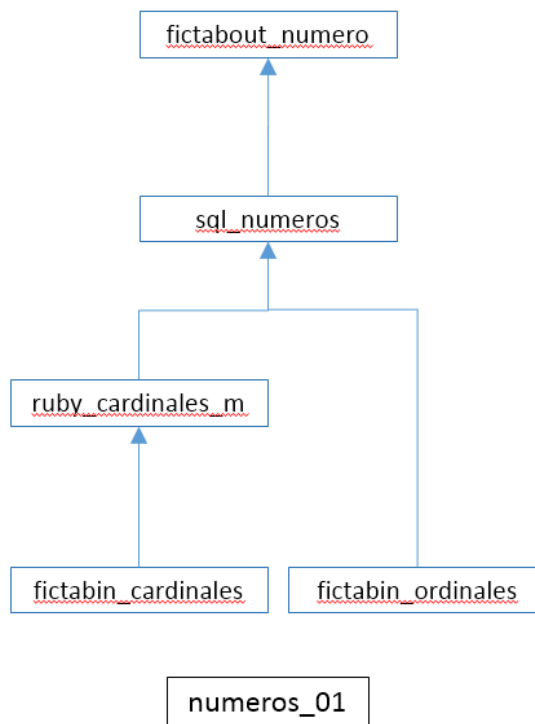


Figura 28: Estructura de procesos elementales de numeros_01

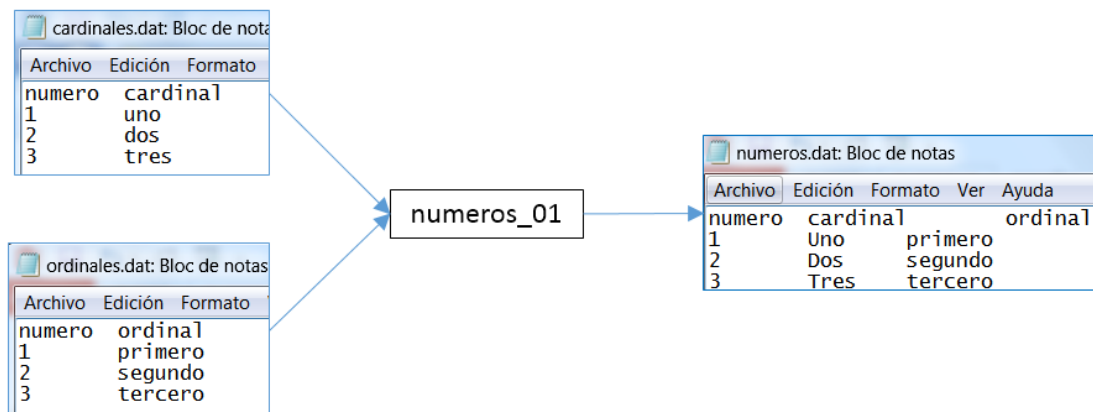


Figura 29: Descripción funcional de numeros_01

numeros_02

Procesos:

1. xltabin_cardinales: importa a WorkDB hoja de cálculo con dos columnas, número y su cardinal.
2. xltabin_ordinales: importa a WorkDB hoja de cálculo con dos columnas, número y su ordinal
3. sql_números: proceso SQL que obtiene una tabla con números, su cardinal y su ordinal.
4. fictabout_numero: exporta la tabla obtenida en el proceso anterior a fichero plano

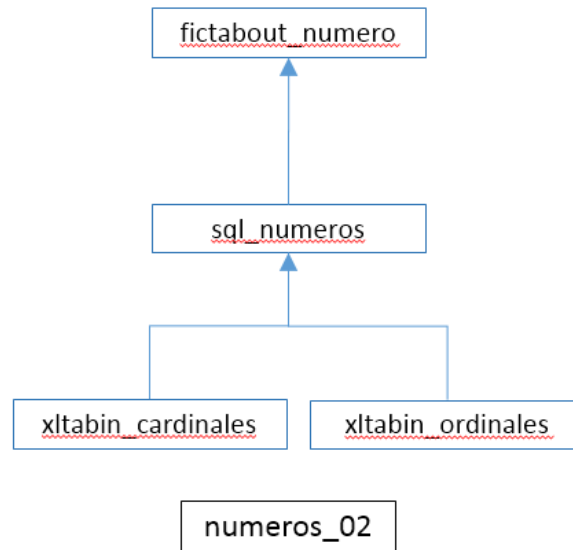


Figura 30: Estructura de procesos elementales de `numeros_02`

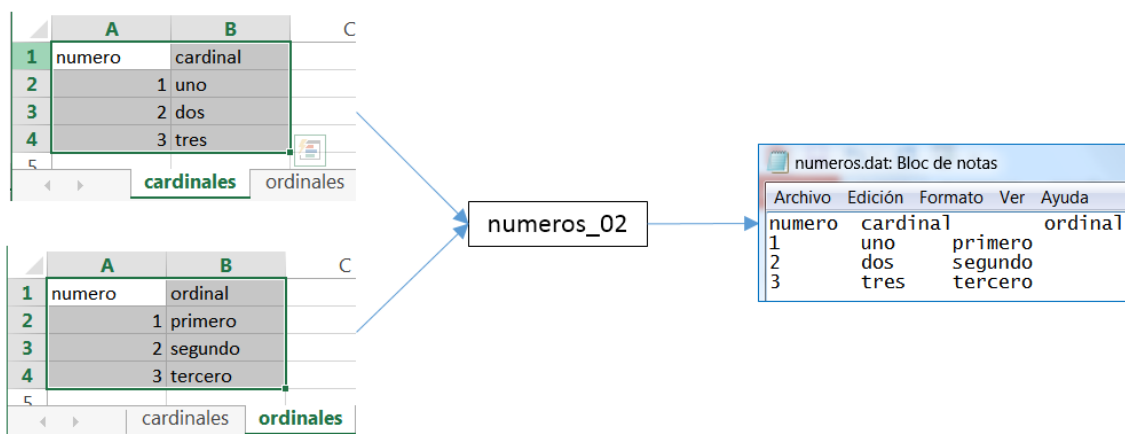


Figura 31: Descripción funcional de `numeros_02`

numeros_03

Procesos:

1. xltabin_cardinales: importa a WorkDB hoja de cálculo con dos columnas, número y su cardinal.
2. xltabin_ordinales: importa a WorkDB hoja de cálculo con dos columnas, número y su ordinal
3. ruby_cardinales_m: proceso ruby que convierte el primer carácter del campo cardinal de la tabla cardinales a mayúscula
4. sql_números: proceso SQL que obtiene una tabla con números, su cardinal y su ordinal.
5. fictabout_numero: exporta la tabla obtenida en el proceso anterior a fichero plano

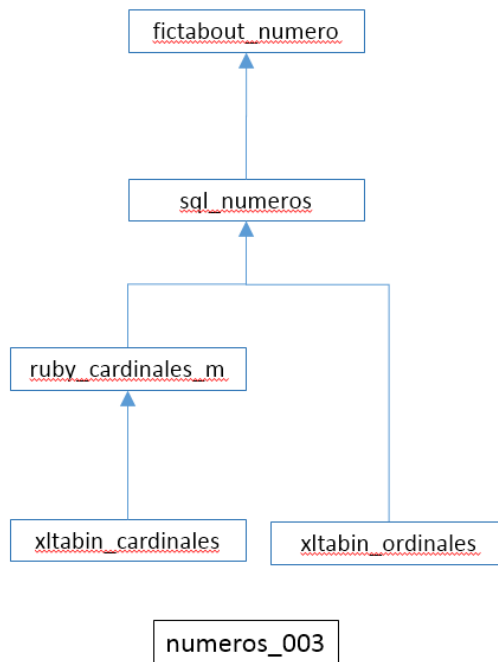


Figura 32: Estructura de procesos elementales de `numeros_03`

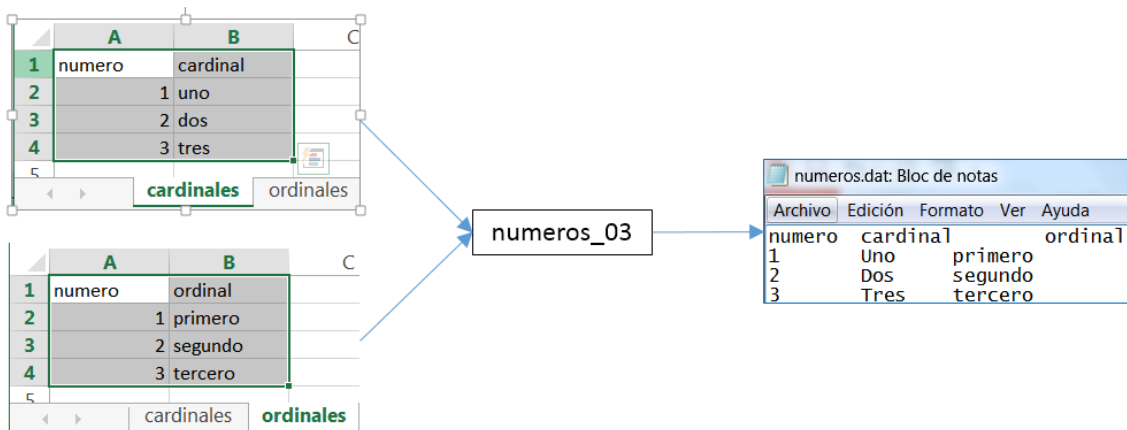


Figura 33: Descripción funcional de `numeros_03`

Base de datos GenDB:

Tenemos cuatro programas y siete procesos relacionados como podemos ver en la figura 34:

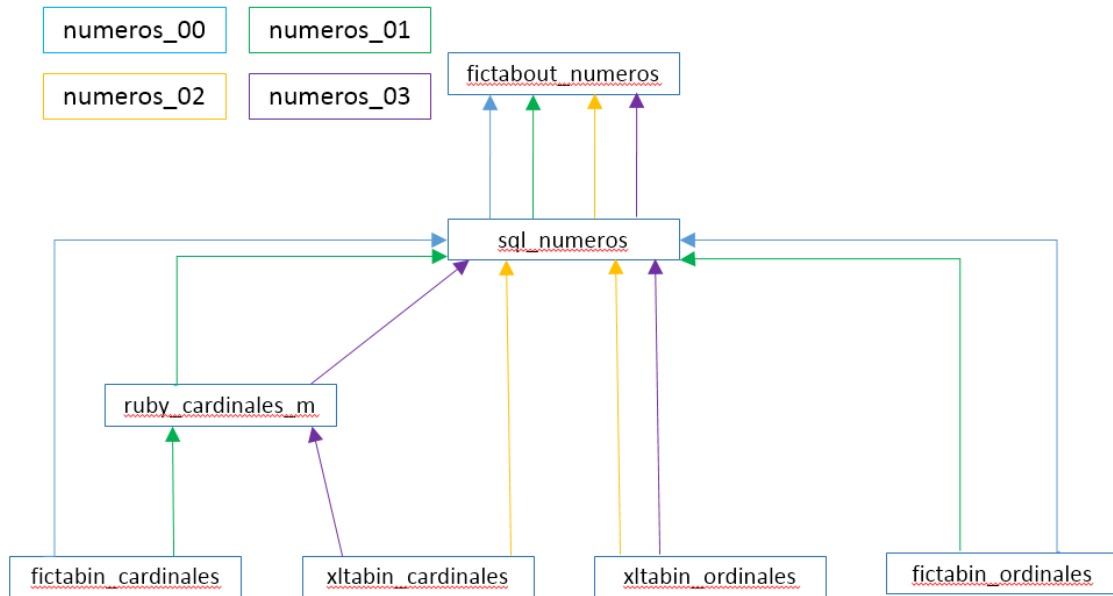


Figura 34: Relación entre los procesos elementales de todos los programas.

El contenido de las tablas para el caso actual sería como sigue.

progmr

prog_no	prog_desc	item_no	prog_path
numeros_00	Obtener números con Cardinal y Ordinal desde fichero.	fictabout_numero	D:_josean_dev\Gen03\numeros
numeros_01	Obtener números con Cardinal y Ordinal desde Excel.	fictabout_numero	D:_josean_dev\Gen03\numeros
numeros_02	Obtener números con Cardinal y Ordinal desde fichero con el primer carácter del cardinal en mayúsculas.	fictabout_numero	D:_josean_dev\Gen03\numeros
numeros_03	Obtener números con Cardinal y Ordinal desde excel con el primer carácter del cardinal en mayúsculas.	fictabout_numero	D:_josean_dev\Gen03\numeros

Tabla 35: Contenido de la tabla progmr del ejemplo 1.

itemmr

item_no	item_desc	item_type	item_code	item_table
fictabin_cardinales	Importa tabla cardinales desde fichero plano	fictabin	cardinales.dat	cardinales
fictabin_ordinales	Importa tabla ordinales desde fichero plano	fictabin	ordinales.dat	ordinales
xltabin_cardinales	Importa tabla cardinales desde excel	xltabin	numeros.xlsx	cardinales
xltabin_ordinales	Importa tabla ordinales desde excel	xltabin	numeros.xlsx	ordinales
ruby_cardinales_m	convierte la primera letra del literal ordinal a mayusculas	ruby	D:_josean_dev\Gen03\numeros\cardinales_m.rb	
sql_numeros	join de cardinal y ordinal por numero	sql	SELECT c.numero, cardinal, ordinal FROM cardinales as c inner join ordinales as o on c.numero = o.numero	numeros
fictabout_numero	Exporta numeros a fichero plano	fictabout	numeros.dat	numeros

Tabla 36: Contenido de la tabla itemmr del ejemplo 1.

prodstru

item_no	item_no_comp	prog_efec
fictabout_numero	sql_numeros	numeros_00
sql_numeros	fictabin_cardinales	numeros_00
sql_numeros	fictabin_ordinales	numeros_00
fictabout_numero	sql_numeros	numeros_01
sql_numeros	xltabin_cardinales	numeros_01
sql_numeros	xltabin_ordinales	numeros_01
fictabout_numero	sql_numeros	numeros_02
sql_numeros	fictabin_ordinales	numeros_02
sql_numeros	ruby_cardinales_m	numeros_02
ruby_cardinales_m	fictabin_cardinales	numeros_02
fictabout_numero	sql_numeros	numeros_03
sql_numeros	xltabin_ordinales	numeros_03
sql_numeros	ruby_cardinales_m	numeros_03
ruby_cardinales_m	xltabin_cardinales	numeros_03

Tabla 37: Contenido de la tabla prodstru del ejemplo 1.

Todos los procesos involucrados a excepción del código ruby son estándar de la aplicación. El código de ruby_cadinales_m es como sigue a continuación:

```
cardinales = @conn_work.from(:cardinales)

cardinales.each do |h|
  cardinales.where( 'numero = ?', h[:numero] ).update( :cardinal => h[:cardinal].capitalize! )
end
```

Conclusión del primer ejercicio

Hemos hecho 4 programas usando 7 procesos elementales, combinándolos en una estructura jerárquica multinivel como se ha visto anteriormente. Los procesos de generación y salida del resultado son compartidos por todos los programas. El proceso de poner la primera letra en mayúscula es opcional y los de importación son compartidos dos a dos. Es la estructura del programa almacenada en la base de datos quien liga todos los procesos para obtener el resultado.

5.2 Caso de estudio con SAP

Para demostrar la integración con SAP ERP vamos a desarrollar tres programas:

1. Consulta al Maestro de Materiales de SAP por los campos “Centro”, “Planificador de necesidades” y “Característica de planificación de necesidades”. Los parámetros se suministran a través de una hoja Excel. Lo llamaremos “MaterialesSAP”.
2. Consulta la Preplanificación en SAP por los campos “centro” y “material”. Los parámetros se suministran a través de una hoja Excel. Lo llamaremos “PreplaMCSAP”.
3. Consulta la Preplanificación en SAP por los campos “centro” y “material”. Los parámetros se suministran a través de la consulta al Maestro de Materiales de SAP citada en primer lugar. Lo llamaremos “PreplaDDWSAP”.

5.2.1 MaterialesSAP

Este programa va a consistir en una consulta a la base de datos de SAP. En particular serán datos de planificación de la tabla MARC (Maestro de materiales, datos específicos de centro) y se accederá por los campos:

- Centro (WERKS)
- Planificador de necesidades (DISPO)
- Característica de planificación de necesidades (DISMM)

Tendrá los siguientes pasos:

1. Importación de los parámetros desde una hoja de cálculo Excel.
2. Conversión en una sentencia Open SQL de ABAP, ejecución y almacenamiento del resultado.
3. Extracción del resultado a un fichero separado por tabulaciones.

Procesos:

1. xltabin_materiales
2. sapsql_materiales
3. fictabout_materiales

Programa que da comienzo en el proceso xltabin_materiales que es una importación desde una hoja Excel. Se obtienen los datos que se van a informar a la tabla sapsql_materiales_parm desde la hoja de cálculo matparm.xlsx.

Con ello se genera la sentencia OpenSQL que se usa para consultar la base de datos de SAP ERP para obtener la información de materiales que se guarda también en la base de datos para luego ser exportada a fichero plano por fictabout_materiales.

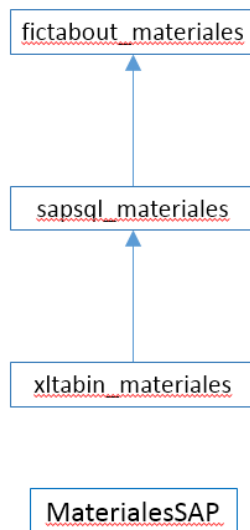


Figura 38: Estructura de procesos elementales de MaterialesSAP

5.2.2 PreplaMCSAP

Este programa es una consulta de los datos de preplanificación en SAP. En particular a las tablas PBIM y PBED que contienen los datos de preplanificación en SAP. Para ello vamos a usar una consulta SELECT de OpenSQL por los siguientes parámetros:

- Centro (WERKS)
- Material (MATNR)

Tendrá los siguientes pasos:

1. Importación de los parámetros desde una hoja de cálculo Excel.
2. Conversión en una sentencia Open SQL de ABAP, ejecución y almacenamiento del resultado.
3. Extracción del resultado a un fichero separado por tabulaciones.

Procesos:

1. xltabin_preplamc
2. sapsql_preplamc
3. fictabout_preplamc

Caso similar al anterior, entrada desde Excel, generación de OpenSQL y obtención de datos para ser exportados en último lugar.

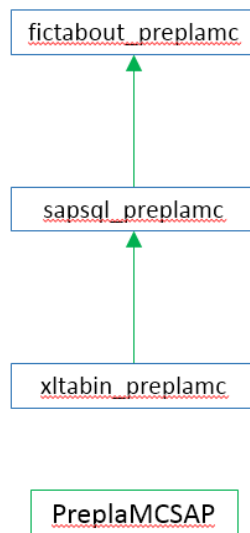


Figura 39: Estructura de procesos elementales de PreplaMCSAP

5.2.3 PreplaDDWSAP

Este programa es una consulta de los datos de preplanificación en SAP pero no es por material como en el caso anterior, sino que usa el parámetro Planificador de necesidades (DISPO) que no se encuentra en las tablas de preplanificación. Pues usaremos la salida del 1er programa MaterialesSAP como entrada de este programa para reutilizar el código. Así que pasaremos los parámetros de MaterialesSAP para que se ejecute y genere la lista de materiales, pero en vez de exportarlo lo vamos a redirigir a la entrada del 2do programa PreplMCSAP para así, reutilizando lo ya hecho anteriormente obtener el resultado.

Tendrá los siguientes pasos:

1. Importación de los parámetros desde una hoja de cálculo Excel.
2. Conversión en una sentencia Open SQL de ABAP, ejecución y almacenamiento del resultado.
3. Conversión de la salida a la entrada del proceso siguiente.
4. Conversión en una sentencia Open SQL de ABAP, ejecución y almacenamiento del resultado.
5. Extracción del resultado a un fichero separado por tabulaciones.

Aquí vamos a ver el resultado del proyecto. Vamos a obtener un programa nuevo reciclando los dos procesos SAP de los programas anteriores. Para ello ejecutaremos uno primero, para obtener los datos de material y centro pero no lo exportaremos sino que mediante una consulta SELECT de SQLite informaremos la tabla de parámetros de entrada del segundo. Después ejecutamos el segundo que es el que exportamos a fichero plano.

Con una simple sentencia SELECT que nos los una hemos podido reciclar los procesos anteriores para obtener un programa con un comportamiento diferente.

Procesos:

1. xltabin_materiales (reutilizado)
2. sapsql_materiales (reutilizado)
3. sql_material_a_preplamc (nuevo)
4. sapsql_preplamc (reutilizado)
5. fictabout_preplamc (reutilizado)

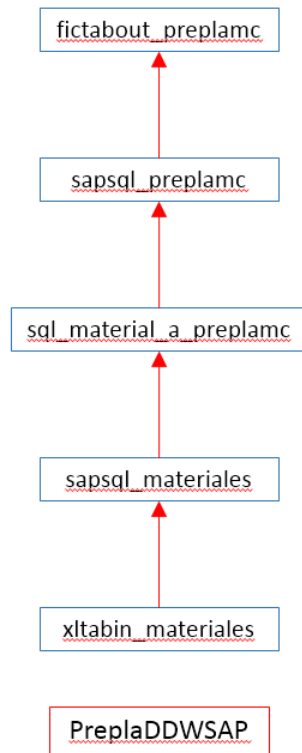


Figura 40: Estructura de procesos elementales de PreplaDDWSAP

En la Figura 40, abajo, se puede apreciar como hay procesos que son compartidos por varios programas. Eso lo conseguimos con el campo *prog_efec*. De este modo dirigimos el sentido de la explosión hacia el proceso correcto.

De forma esquemática podemos ver los flujos en el diagrama de abajo y ver como se comparten.

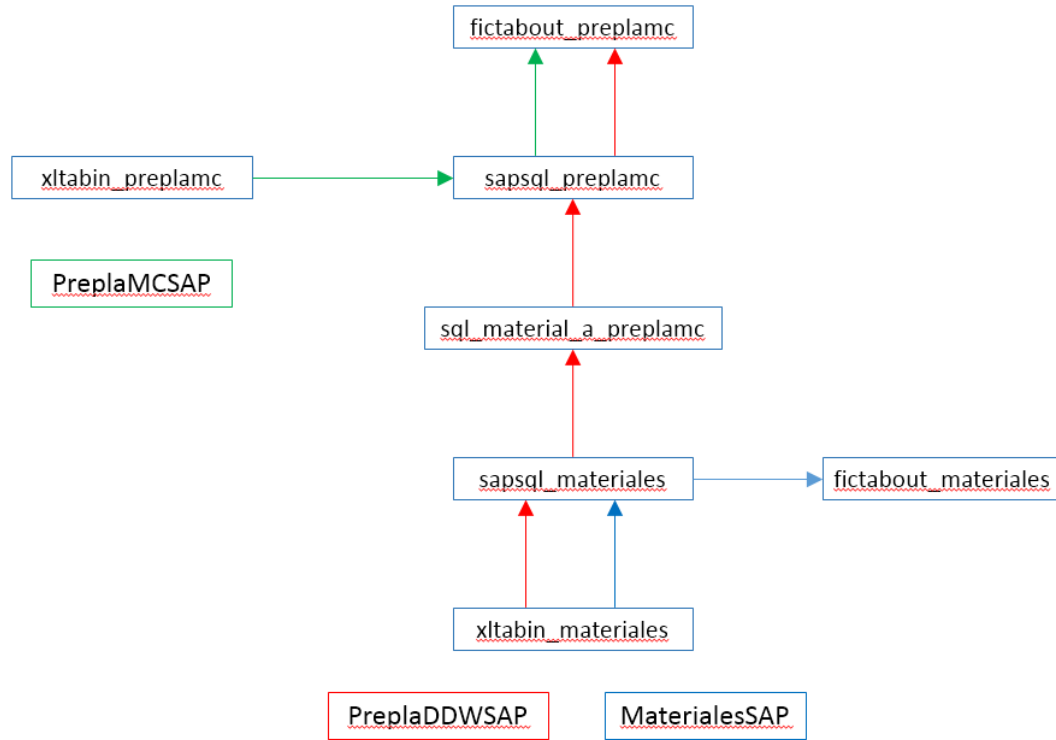


Figura 41: Flujo en la base de datos del ejemplo 2.

Hemos definido, por lo tanto tres programas tal y como informamos en la tabla progmsr:

prog_no	prog_desc	item_no	prog_path
MaterialesSAP	Materiales por WERKS, DISPO y DISMM desde SAP	fictabout_materiales	D:_josean_prog\MaterialesSAP
PreplaMCSAP	Preplanificación descargada de SAP por MATNR_EXT y WERKS suministrado por excel.	fictabout_preplamc	D:_josean_prog\PreplaMCSAP
PreplaDDWSAP	Preplanificación descargada de SAP por MATNR_EXT y WERKS suministrado desde una consulta a SAP por WERKS, DISPO y DISMM.	fictabout_preplamc	D:_josean_prog\PreplaDDWSAP

Tabla 42: Contenido de la tabla progmsr del ejemplo 2.

Hemos definido 7 procesos elementales basados en nuestras plantillas como se puede ver más abajo en el contenido de la tabla itemmstr

Y por último establecemos las relaciones jerárquicas en la tabla prodstru:

item_no	item_no_comp	prog_efec
fictabout_materiales	sapsql_materiales	
sapsql_materiales	xltabin_materiales	
fictabout_preplamc	sapsql_preplamc	
sapsql_preplamc	xltabin_preplamc	PreplaMCSAP
sapsql_preplamc	sql_material_a_preplamc	PreplaDDWSAP
sql_material_a_preplamc	sapsql_materiales	

Tabla 43: Contenido de la tabla prodstru del ejemplo 2.

item_no	item_desc	item_type	item_code	item_table
sapsql_materiales	SAP Consulta Maestro de Materiales	sapsql	SELECT MI~MATNR_EXT MC~MATNR MT~MAKTX MC~WERKS MC~DISPO MC~DISMM MC~BESKZ FROM MATERIALID AS MI INNER JOIN MAKT AS MT ON MI~MATNR_INT = MT~MATNR INNER JOIN MARC AS MC ON MI~MATNR_INT = MC~MATNR FOR ALL ENTRIES IN sapsql_materiales_parm WHERE MT~SPRAS EQ 'S' AND MC~WERKS EQ sapsql_materiales_parm-werks AND MC~DISPO EQ sapsql_materiales_parm-dispo AND MC~DISMM EQ sapsql_materiales_parm-dismm	sapsql_materiales
xltabin_materiales	Fichero de parámetros para obtener Materiales de SAP	xltabin	matparm.xlsx	sapsql_materiales_parm
xltabin_preplamc	Popular sapsql_preplamc_parm	xltabin	matcen.xlsx	sapsql_preplamc_parm
sapsql_preplamc	Obtener Preplanificación por Material Centro de SAP	sapsql	select PBIM~PBDNR PBIM~WERKS PBIM~MATNR PBIM~BEDAE PBIM~VERSB PBIM~PBDNR PBIM~ZUVKZ PBIM~VERVS PBIM~LOEVR PBIM~BDZEI PBIM~VERKZ PBIM~ZUOKR PBIM~DATLP PBIM~UHRLP PBIM~AGGPB PBED~PDATU PBED~AENAM PBED~LAEDA PBED~MEINS PBED~PLNMG PBED~ENTLU PBED~PERXX PBED~WDATU PBED~ENTMG PBED~UPLMG PBED~ENTLI PBED~BDZEI from (PBIM inner join PBED on PBED~BDZEI = PBIM~BDZEI) for all entries in sapsql_preplamc_parm where PBIM~WERKS eq sapsql_preplamc_parm-werks and PBIM~PBDNR eq sapsql_preplamc_parm-matnr	sapsql_preplamc
fictabout_materiales	Extraccion de Materiales desde SAP a FicSepTab	fictabout	materiales.txt	sapsql_materiales
fictabout_preplamc	Exportar Preplanificación pro Material y Centro desde SAP	fictabout	preplamc.txt	sapsql_preplamc
sql_material_a_preplamc	Convertir la tabla generada por sapsql_materiales en el fichero de parámetros para sapsql_preplamc	sql	select matnr_ext, werks from sapsql_materiales	sapsql_preplamc_parm

Tabla 44: Contenido de la tabla itemmstr del ejemplo 2.

6. Conclusiones y líneas futuras

Este trabajo pretendía explorar las posibilidades de aplicar Ingeniería de Configuración de materiales al Desarrollo y la Generación de Software. Obviamente este trabajo puede desembocar en una aplicación compleja que dé servicio a un grupo de trabajo y que sirva tanto para tener clasificada la producción de software como para, y esto es lo más importante a mi entender, fomentar su reutilización.

Es por ello que de cara el futuro se puede usar este trabajo como base y trabajar en las siguientes líneas:

- Integración en una aplicación de escritorio que permita mantener los programas, procesos y sus relaciones.
- Mejora de la RFC de comunicación con SAP, Z_RFC_SQL:
 - Incluir chequeos de seguridad de datos
 - Incluir más recursos de SQL
- Uso del diccionario de SAP para obtener información de tipo y encabezamiento de los campos.
- Despliegue basado en servidor para dar servicio a un equipo
- Mejora en las características de clasificación de los programas para poder agrupar por módulo y para poder tener identificadas las tablas a las que se accede en SAP
- Desarrollo de más módulos como por ejemplo:
 - Exportar tabla a Excel via Ole
 - Exportar tabla a Excel via formato XML de Excel
 - Importar tabla desde bases de datos.
 - Web services
- Mejorar la efectividad para poder tener efectividades más complejas, no solo por programa, sino también por conjuntos de programas.

Apéndices

A1 – Programa Desarrollador

```
# Desarrollador.rb
# Author:: Jose Antonio Reguera
#
# Programa de ayuda para depurar los datos incluidos en la base de
# Datos GenDB.
# El resultado final será el mismo que el programa que se generará más
# tarde con
# el programa Generador.rb.
# * Conexión a GenDB.
# * Conexión a WorkDB.
# * Obtención de la secuencia de ejecución y almacenado en WorkDB si
# no existe.
# * Búsqueda de la primera entrada de la secuencia que no se ha
# completado.
# * Ejecución de dicho proceso.
# * Actualización de la secuencia con el flag de procesado.
# * Si queda algun proceso ir al paso anterior de búsqueda y si no
# informar de la conclusión.

# libreria para JDBC SQLite3
require 'C:/japath/sqlite-jdbc-3.8.10.1.jar'
# libreria e includes para JCo3 de SAP
require 'C:/sapjco/sapjco3.jar'
java_import com.sap.conn.jco.JCoTable
java_import com.sap.conn.jco.JCoDestinationManager
java_import com.sap.conn.jco.JCoContext

# Gems necesarias para la ejecucion
require 'nokogiri'
require 'jdbc/sqlite3'
require 'win32ole'
require 'sequel'

# Clase principal
class Desarrollador

  # Metodo de entrada en la aplicacion.
  def ejec_programa programa
    # Conexion a GenDB.
    @programa = programa
    db_gendata = "#{__dir__}\\GenData.sqlite"
    puts "Path a Gendata:#{db_gendata}"
    @conn_gen = Sequel.connect "jdbc:sqlite:#{db_gendata}"
    # Ubicacion para cuando se tienen distintos directorios en
    desarrollo.
    @ubicacion = @conn_gen.from(:progmstr).where(:prog_no =>
programa).first[:prog_path]
    puts "Ubicacion:#{@ubicacion}"
    Dir.chdir @ubicacion
    # Conexion a WorkDB.
    db_work = "#{@ubicacion}\\work.sqlite"
    @conn_work = Sequel.connect "jdbc:sqlite:#{db_work}"
    # Sacamos la secuencia a ejecutar.
    calc_sec @conn_gen.from(:progmstr).where(:prog_no =>
programa).first[:item_no]
    # Ejecutamos la secuencia obtenida.
```

```

@conn_work[:__secuencia__].where(:status=>false).reverse_order(:sequence).each do |row|
  ejec_proc row[:item_no]
end
@conn_work.disconnect
@conn_gen.disconnect
# Informamos del fin de la ejecucion.
puts 'That''s all folks!!!'
end

private

# Obtenemos la secuencia de procesos a ejecutar.
def calc_sec proceso
  # Comprobamos si existe secuencia en WorkDB.
  unless @conn_work.table_exists? :__secuencia__ then
    # Creamos la tabla para alojar la secuencia.
    @conn_work.create_table :__secuencia__ do
      column :item_no, :text
      column :sequence, :integer
      column :status, :boolean
    end
    # Calculamos secuencia de ejecución.
    func_rec( proceso, 0 )
    puts "Comenzamos encontrando la secuencia de
ejecucion."
    puts '<----->'
    @conn_work[:__secuencia__].each do |row|
      puts row
    end
    puts '<----->'
  end
end

# Creamos tabla en WorkDB.
def crea_tabla item_table, campos, filas
  # Creamos la tabla.
  puts "Creamos la tabla:#{item_table}"
  @conn_work.create_table item_table.to_sym do
    campos.each do |c|
      send( 'COLUMN', c.to_sym )
    end
  end
  # Insertamos los datos en la tabla recién creada en WorkDB.
  @conn_work[item_table.to_sym].import campos, filas
end

# Ejecutamos proceso elemental.
def ejec_proc item_no
  puts "Ejecutamos '#{item_no}'."
  item_type = ''
  item_code = ''
  # Identificamos el tipo de proceso.
  row = @conn_gen.from(:itemmstr).where(:item_no =>
item_no).first
  item_type = row[:item_type]
  item_code = row[:item_code]
  item_table = row[:item_table]
  # Ejecutamos el proceso según su tipo.
  puts "Es un proceso de tipo '#{item_type}'."
end

```

```

    if item_code.empty? then
      puts "Proceso '#{item_no}' vacio. Saliendo..."
      abort
    end
    case item_type
      when 'ruby'
        proc_ruby item_no, item_code, item_table
      when 'sql'
        proc_sql item_no, item_code, item_table
      when 'fictabin'
        proc_fictabin item_no, item_code, item_table
      when 'fictabout'
        proc_fictabout item_no, item_code, item_table
      when 'sapsql'
        proc_sapsql item_no, item_code, item_table
      when 'xltabin'
        proc_xltabin item_no, item_code, item_table
      else
        puts "Tipo de proceso '#{item_type}' invalido!"
        abort
      end
    puts "Proceso #{item_no} Ok."
    # Informamos de que el proceso ha sido ejecutado.
    @conn_work[:__secuencia__].where(:item_no =>
item_no).update(:status => true)
    end # method

    # Obtencion de la secuencia de ejecución a partir de GenDB con
    un algoritmo recursivo.
    def func_rec mi_proceso, nivel
      if @conn_work[:__secuencia__].where( :item_no => mi_proceso
).count != 0 then
        # Evitamos bucles en la recursividad.
        puts "Proceso #{mi_proceso} recursivo!"
        abort
      else
        # Actualizamos la secuencia de ejecución
        @conn_work[:__secuencia__].insert(:item_no => mi_proceso,
:sequence => nivel, :status => false)
        # Continuamos la recursividad.
        @conn_gen.fetch "SELECT item_no_comp FROM prodstru
WHERE item_no = '#{mi_proceso}' and (prog_efec is null or prog_efec =
'#{@programa}')" do |row|
          func_rec row[:item_no_comp], nivel + 1
        end
      end
    end

    # Proceso 'fictabin'. Importamos a WorkDB de una tabla desde un
    fichero plano
    # separado por tabulaciones y con los nombres de los campos en
    la primera fila.
    def proc_fictabin item_no, item_code, item_table
      # Leemos el fichero plano.
      if File.file?( item_code ) then
        fichero = File.read( item_code )
      else
        puts "El fichero '#{item_code}' no existe!"
        exit
      end
    end

```

```

        # Populamos las filas en un Array que a su vez hemos
convertido en un Array de campos.
        filas = fichero.split("\n").collect { |d| d.split("\t") }
        # Sacamos la primera fila que es donde se encuentran los
nombres de campo.
        campos = filas.shift
        # Creamos la tabla.
        crea_tabla item_table, campos, filas
        # Informo del fin del proceso elemental.
        puts "Tabla '#{item_table}' creada!"
    end

    # Proceso 'fictabout'. Exportamos una tabla a un fichero plano
separado por
    # tabulaciones con los nombres de campos en la primera fila.
    def proc_fictabout item_no, item_code, item_table
        fl_path = item_code
        # puts fl_path
        if File.file?(fl_path) then
            # Si existe informamos y borramos.
            puts "Fichero '#{fl_path}' existe. Borrado!"
            File.delete(fl_path)
        end
        # Creamos el fichero y lo populamos desde la tabla.
        fl = File.new("#{fl_path}", 'w')
        ds = @conn_work.from(item_table.to_sym)
        fl.puts "#{ds.columns.join("\t")}\r"
        fl.puts
            "#{ds.map(ds.columns).collect{|a|
a.join("\t")}.join("\r\n")}\r"
        fl.close
        # Informo del fin del proceso elemental.
        puts "Fichero '#{fl.path}' creado!"
    end

    # Proceso 'ruby'. Se ejecuta código aleatorio ruby.
    def proc_ruby item_no, item_code, item_table
        # Si es un fichero del disco duro le informamos en item_code.
        if item_code.lines.count == 1 then
            if File.file?( item_code ) then
                puts "Existe externo."
                item_code = File.read( item_code )
            end
        end
        # Evaluemos el código.
        eval item_code
        # Informo del fin del proceso elemental.
        puts "Proceso ruby #{item_no} ejecutado!"
    end

    # Proceso 'sapsql'. Ejecuta una sentencia SELECT de SAP Open SQL
en item_code informando del resultado
    # en una tabla de nombre item_table en WorkDB
    def proc_sapsql item_no, item_code, item_table

        # Regex para parsear la sentencia SELECT.
        regex_groupby = /(.*)\bgroup by\b(.*)/im
        regex_limit = /(.*)\blimit\b(.*)/im
        regex_orderby = /(.*)\border by\b(.*)/im
        regex_tabla = /(.*)\bfor all entries in (\w+)/im
        regex_where = /where (.*)/im
    end

```



```

# Obtenemos la clausula SELECT.
select = item_code.downcase
puts "Select:#{select}"
tabla = ''
antes_del_where = ''
clausula_where = ''
resto_clausulas = ''

# Obtenemos la tabla de WorkDB con los parámetros para
generar la clausula WHERE
# y obtenemos la clausula WHERE.
tmatch_tabla = select.match regex_tabla
if tmatch_tabla then
  antes_del_where = tmatch_tabla[1]
  tabla = tmatch_tabla[2]
else
  puts "Vaya! No encuentro tabla :-(
  abort
end

puts "Antes del where:#{antes_del_where}"
puts "Tabla:#{tabla}"

regex_campos = /\b(#{tabla}\-\S+)\b/im

tmatch_where = select.match regex_where
if tmatch_where then
  clausula_where = tmatch_where[1]
end

puts "Clausula where:#{clausula_where}"

# Obtenemos el resto de las clausulas.
tmatch_orderby = clausula_where.match regex_orderby
if tmatch_orderby then
  clausula_where = tmatch_orderby[1]
  resto_clausulas = " order by #{tmatch_orderby[2]}"
else
  tmatch_groupby = clausula_where.match regex_groupby
  if tmatch_groupby then
    clausula_where = tmatch_groupby[1]
    resto_clausulas = "          group      by
#{tmatch_groupby[2]}"
  else
    tmatch_limit = clausula_where.match regex_limit
    if tmatch_limit then
      clausula_where = tmatch_limit[1]
      resto_clausulas = "          limit
#{tmatch_limit[2]}"
    end
  end
end

puts "Resto clausulas:#{resto_clausulas}"

# Construimos la sentencia WHERE nueva con los datos de la
tabla de paramtros.
campos_where = clausula_where.scan regex_campos

# creamos un dataset de la tabla de 'for all entries'
items = @conn_work[tabla.to_sym]

```

```

witems = Array.new
items.each do |i|
  witem = "( #{clausula_where.chomp} )"
  campos_where.each do |c|
    campo = c.first
    n = campo.gsub("#{tabla}-", '')
    witem.gsub!(campo, "'#{i[n.to_sym]}'")
  end
  witems.push witem
end

# Construimos la nueva sentencia SELECT que mandaremos a
SAP ERP.
item_code = "#{antes_del_where} where #{witems.join("\nor
")} #{resto_clausulas}".upcase
puts item_code

# Iniciamos soporte RFC SAP
campos = Array.new
filas = Array.new
$sald = 'ALD'
destination = JCoDestinationManager.getDestination($sald)
JCoContext.begin destination
# Funcion Z RFC SQL
function = destination.getRepository.getFunction("Z RFC SQL")
tsqltab = function.getTableParameterList().getTable("TSQLTAB")
# Informamos de la sentencia SELECT generada.
item_code.each_line do |l|
  tsqltab.addRow()
  tsqltab.setValue('ZLINEA', l.chomp)
end

# Ejecutamos la RFC
begin
  function.execute(destination)
rescue => mensaje
  puts "Error: #{mensaje.to_s}"
  return
end

# Parseamos el XML resultado.
txmldata = function.getTableParameterList().getTable("TXMLDATA")
s_xml = ''
txmldata.firstRow()
begin
  xmldata = txmldata.getValue('ZLINEA')
  s_xml << xmldata
end while txmldata.nextRow()
s_xml = s_xml[s_xml.index('<asx:abap'),s_xml.length-
s_xml.index('<asx:abap')+1]
doc = Nokogiri::XML(s_xml)
doc.xpath("//item").each_with_index {|rec, index|
  fila = Array.new
  rec.children.each {|campo|
    if index == 1 then
      campos.push campo.name.downcase
    end
    fila.push campo.content
  }
  filas.push fila
}

# Creamos la tabla en WorkDB.

```

```

        crea_tabla item_table, campos, filas
        # Informo del fin del proceso elemental.
        puts "Proceso sapsql #{item_no} ejecutado!"
    end

    # Proceso 'sql'. Creo una tabla en WorkDB con nombre item_table a
    # partir de una sentencia
    # SELECT de SQLite3 en item_code.
    def proc_sql item_no, item_code, item_table
        @conn_work.run "create table #{item_table} as #{item_code}"
        # Informo del fin del proceso elemental.
        puts "Proceso sql #{item_no} ejecutado!"
    end

    # Proceso 'xltabin'. Importamos una tabla en WorkDB con nombre
    # en item_table
    # a partir de una hoja de Microsoft Excel via OLE. Nombres de
    # campo en la primera fila.
    def proc_xltabin item_no, item_code, item_table
        # Ubicacion del libro Microsoft Excel en item_code.
        xlpath = item_code
        # Nombre de la hoja a importar en item_table.
        xltable = item_table
        puts "Proceso xltabin; Path:#{xlpath} Table:#{xltable}"
        # Conectamos a WIN32OLE.
        begin
            excel_app = WIN32OLE.connect("excel.application")
        rescue
            excel_app = WIN32OLE.new("excel.application")
        end
        puts "Excel default file path:#{excel_app.DefaultFilePath}"
        excel_app.DefaultFilePath = @ubicacion
        excel_app.DisplayAlerts = false
        # Abrimos libro e identificamos hoja.
        wkb = excel_app.workbooks.open xlpath
        wsh = wkb.sheets(xltable)
        # Obtenemos rango usado.
        rango = wsh.usedrange
        filas = Array.new
        # Populamos un Array
        rango.rows.each { |r|
            columnas = Array.new
            r.columns.each { |c|
                columnas.push c.value
            }
            filas.push columnas
        }
        rango = nil
        wsh = nil
        wkb.Close#(0)
        wkb = nil
        excel_app.Quit
        excel_app.ole_free
        excel_app = nil
        GC.start
        unless filas.length > 1 then
            puts "No hay datos!"
            abort
        end
        # Obtenemos campos de la primera fila.
        campos = filas.shift
    end

```

```

        # Creamos la tabla en WorkDB.
        crea_tabla item_table, campos, filas
        # Informo del fin del proceso elemental.
        puts "Proceso xltabin {item_no} ejecutado!"
    end

end # class

# main. Punto de entrada del programa.

proc = Desarrollador.new()

begin
    proc.ejec_programa 'PreplaDDWSAP'
rescue => problem
    print "{problem}\n"
    exit
end

```

A2 – Programa Generador

```
# Generador
# Generador.rb
# Author:: Jose Antonio Reguera
#
# Programa que a partir de los datos incluidos en la base de Datos
# GenDB, genera un programa
# independiente.
# El programa generado tiene una estructura con un método principal en
# la parte pública
# que conecta a la base de datos WorkDB y la secuencia de procesos que
# se ejecutarán.
# En la parte privada tiene los métodos que corresponden a los
# procesos generados.
# El proceso de generación recibe como parámetro el programa a
# generar.
# Son sus etapas:
# * Conexión a GenDB.
# * Obtención por un proceso recursivo de la secuencia de ejecución y
# almacenado en un array.
# * Para la parte principal se itera sobre el array para obtener la
# secuencia de los procesos a ejecutar.
# * Acto seguido se vuelve a iterar sobre el array de procesos
# elementales, por cada elemento
# * se identifica y mediante una plantilla ERB por tipo de proceso se
# genera el código de esta parte.
# * Para finalizar una plantilla ERB lo une todo formando el programa
# generado.
```

```
require 'C:/japath/sqlite-jdbc-3.8.10.1.jar'
require 'jdbc/sqlite3'
require 'erb'
require 'sequel'
```

class Generador

```
def genera programa
  salida = ''
  principal_code = ''
  funciones_code = ''
  db_gendata = "#{__dir__}\\GenData.sqlite"
  puts "Path a Gendata:#{db_gendata}"
  # Conexión a GenDB.
  @conn_gen = Sequel.connect "jdbc:sqlite:#{db_gendata}"
  # Sacamos la secuencia a ejecutar.
  resultado = Hash.new
  proceso_raiz = @conn_gen.from(:progmsr).where(:prog_no =>
programa).first[:item_no]
  func_rec( proceso_raiz, resultado, 0 )
  miarr = resultado.to_a.sort_by{|k,v| v}.reverse
  puts "Comenzamos encontrando la secuencia de ejecución."
  puts '<----->'
  miarr.each { |a| puts "#{a[0]}, nivel #{a[1]}" }
  puts '<----->'
  # Obtenemos el código de la secuencia de procesos.
  principal_code = miarr.map{|e| e[0]}.join("\r\n")
  puts principal_code
  funciones_code = proceso_funciones miarr
  puts funciones_code
  # Obtenemos el programa generado.
```

```

begin
  fh = File.new( "templates/template_main.rb" )
  erb_script = fh.read
  fh.close()
rescue
  raise "No encuentro plantilla main!"
end

begin
  t = ERB.new( erb_script )
  salida = t.result( binding )
rescue => err
  raise err #, "There was a problem interpreting
template #{content['Template']}"
end

begin
  File.write( "Salida.rb", salida )
rescue
  raise "Could not create fichero de salida"
end

puts 'That''s all folks!!!'

end

private

# Obtención de la secuencia de procesos.
def func_rec mi_proceso, resultado, nivel

  if resultado.has_key?(mi_proceso)
    puts "Proceso #{mi_proceso} bucle recursivo!"
    abort
  else
    resultado[mi_proceso] = nivel # añadimos
    @conn_gen.fetch "SELECT item_no_comp FROM prodstru
WHERE item_no = '#{mi_proceso}' and (prog_efec is null or prog_efec =
 '#{@programa}')" do |row|
      func_rec row[:item_no_comp], resultado, nivel +
1
    end
  end
end

# Generación de los métodos correspondientes a los tipos de
proceso con las plantillas ERB.
def proceso_funciones miarr
  salida = ''
  miarr.each do |ar|
    mi_funcion = ''
    item_no = ar[0]
    ds_im = @conn_gen.fetch("SELECT * FROM itemmstr WHERE item_no =
 '#{item_no}").first
    item_type = ds_im[:item_type]
    item_code = ds_im[:item_code]
    item_table = ds_im[:item_table]
    # Generamos el proceso.
    puts "Es un proceso de tipo '#{item_type}'."
    if item_code.empty? then
      puts "Proceso '#{item_no}' vacio. Saliendo..."
    end
  end
end

```

```

    exit
  end
  if item_type == 'ruby' then
    if item_code.lines.count == 1 then
      if File.file?( item_code ) then
        puts "Existe externo."
        item_code = File.read( item_code )
      end
    end
  end
  end
  # Generamos erb.
  item_code.gsub!("\n","\n\n")
  begin
    fh = File.new( "templates/template_#{item_type}.rb" )
    erb_script = fh.read
    fh.close()
  rescue
    raise "No encuentro plantilla template_#{item_type}.rb!"
  end
  begin
    t = ERB.new( erb_script )
    mi_funcion = t.result( binding )
  rescue => err
    raise err #, "There was a problem interpreting template
#{content['Template']}"
  end
  salida << mi_funcion
  puts "Proceso #{item_no} Ok."
end
salida
end

end

Dir.chdir __dir__
puts __dir__

proc = Generador.new()
begin
  proc.genera 'MaterialesSAP'
rescue => problem
  print "#{problem}\n"
  exit
end
end

```

A3 – Templates del generador

template_fictabin.rb

```
def <%= item_no %>

    item_code = '<%= item_code %>'
    item_table = '<%= item_table %>'
    if File.file?( item_code ) then
        fichero = File.read( item_code )
    else
        puts "El fichero '#{item_code}' no existe!"
        exit
    end
    filas = fichero.split("\n").collect { |d| d.split("\t") }
    campos = filas.shift
    # creamos la tabla
    crea_tabla item_table, campos, filas

end
```

template_fictabout.rb

```
def <%= item_no %>

    fl_path = '<%= item_code %>'
    item_table = '<%= item_table %>'
    # puts fl_path
    if File.file?(fl_path) then
        puts "Fichero #{fl_path} existe. Borrado!"
        File.delete(fl_path)
    end
    fl = File.new("#{fl_path}", 'w')
    puts "Fichero #{fl.path} creado!"
    ds = @conn_work.from(item_table.to_sym)
    fl.puts "#{ds.columns.join("\t")}\r"
    fl.puts          "#{ds.map(ds.columns).collect{|a|
a.join("\t")}.join("\r\n")}\r"
    fl.close

end
```

template_main.rb

```
require 'C:/japath/sqlite-jdbc-3.8.10.1.jar'
require 'C:/sapjco/sapjco3.jar'
java_import com.sap.conn.jco.JCoTable
java_import com.sap.conn.jco.JCoDestinationManager
java_import com.sap.conn.jco.JCoContext

require 'nokogiri'
require 'jdbc/sqlite3'
require 'win32ole'
require 'sequel'

class Generada

    def proceso
```



```

# Conexión a la base de datos
@conn_work = java.sql.DriverManager.getConnection
'jdbc:sqlite:work.sqlite'

# inicializamos tablas y secuencia de procesos
<%= principal_code %>

# Finalizams
puts 'That''s all folks!!!'
@conn_work.close
end

private

def crea_tabla item_table, campos, filas
  # creamos la tabla
  puts "Creamos la tabla:#{item_table}"
  @conn_work.create_table item_table.to_sym do
    campos.each do |c|
      send( 'COLUMN', c.to_sym )
    end
  end
  # insertamos los datos
  @conn_work[item_table.to_sym].import campos, filas
end

<%= funciones_code %>

end

proc = Generada.new()
begin
  proc.proceso
rescue => problem
  print "#{problem}\n"
  abort
end

```

template_ruby.rb

```

def <%= item_no %>
  <%= item_code %>

end

```

template_sapsql.rb

```

def <%= item_no %>

  item_code = "<%= item_code %>"
  item_table = "<%= item_table %>"
  regex_groupby = /(.*)\bgroup by\b(.*)/im
  regex_limit = /(.*)\blimit\b(.*)/im
  regex_orderby = /(.*)\border by\b(.*)/im
  regex_tabla = /(.*)\bfor all entries in (\w+)/im
  regex_where = /where (.*)/im

  select = item_code.downcase

```

```

# puts "Select:#{select}"
tabla = ''
antes_del_where = ''
clausula_where = ''
resto_clausulas = ''

tmatch_tabla = select.match regex_tabla
if tmatch_tabla then
  antes_del_where = tmatch_tabla[1]
  tabla = tmatch_tabla[2]
else
  puts "Vaya! No encuentro tabla :-(
  abort
end

puts "Antes del where:#{antes_del_where}"
puts "Tabla:#{tabla}"

regex_campos = /\b(#{tabla}\-\S+)\b/im

tmatch_where = select.match regex_where
if tmatch_where then
  clausula_where = tmatch_where[1]
end

puts "Clausula where:#{clausula_where}"

tmatch_orderby = clausula_where.match regex_orderby
if tmatch_orderby then
  clausula_where = tmatch_orderby[1]
  resto_clausulas = " order by #{tmatch_orderby[2]}"
else
  tmatch_groupby = clausula_where.match regex_groupby
  if tmatch_groupby then
    clausula_where = tmatch_groupby[1]
    resto_clausulas = " group by
#{tmatch_groupby[2]}"
  else
    tmatch_limit = clausula_where.match regex_limit
    if tmatch_limit then
      clausula_where = tmatch_limit[1]
      resto_clausulas = " limit
#{tmatch_limit[2]}"
    end
  end
end

puts "Resto clausulas:#{resto_clausulas}"

campos_where = clausula_where.scan regex_campos

# creamos un dataset de la tabla de 'for all entries'
items = @conn_work[tabla.to_sym]

witems = Array.new
items.each do |i|
  witem = "( #{clausula_where.chomp} )"
  campos_where.each do |c|
    campo = c.first
    n = campo.gsub("#{tabla}-", '')
    witem.gsub!(campo, "'#{i[n.to_sym]}'")
  end
end

```

```

        end
        witems.push witem
    end

    item_code = "#{antes_del_where} where #{witems.join("\nor
")} #{resto_clausulas}".upcase
    puts item_code

# empezamos con la parte SAP

    campos = Array.new
    filas = Array.new
    $ald = 'ALD'
    destination = JCoDestinationManager.getDestination($ald)
    JCoContext.begin destination
    function = destination.getRepository.getFunction("Z_RFC_SQL")
    tsqldata = function.getTableParameterList().getTable("TSQLTAB")
    item_code.each_line do |l|
        tsqldata.addRow()
        tsqldata.setValue('ZLINEA',l.chomp)
    end
    begin
        function.execute(destination)
    rescue => mensaje
        puts "Error: #{mensaje.to_s}"
        return
    end
    txmldata = function.getTableParameterList().getTable("TXMLDATA")
    s_xml = ''
    txmldata.firstRow()
    begin
        xmldata = txmldata.getValue('ZLINEA')
        s_xml << xmldata
    end while txmldata.nextRow()
    s_xml = s_xml[s_xml.index('<asx:abap'),s_xml.length-
s_xml.index('<asx:abap')+1]
    doc = Nokogiri::XML(s_xml)
    doc.xpath("//item").each_with_index {|rec, index|
        fila = Array.new
        rec.children.each {|campo|
            if index == 1 then
                campos.push campo.name.downcase
            end
            fila.push campo.content
        }
        filas.push fila
    }
    # creamos la tabla
    crea_tabla item_table, campos, filas
end

```

template_sql.rb

```

def <%= item_no %>

    item_code = "<%= item_code %>"
    item_table = "<%= item_table %>"
    @conn_work.run "create table #{item_table} as #{item_code}"

end

```

template_xltabin.rb

```
def <%= item_no %>

  xlpath = "<%= item_code %>"
  xltable = "<%= item_table %>"
  puts "Proceso xltabin; Path:#{xlpath} Table:#{xltable}"
  begin
    excel_app = WIN32OLE.connect("excel.application")
  rescue
    excel_app = WIN32OLE.new("excel.application")
  end
  puts "Excel default file path:#{excel_app.DefaultFilePath}"
  excel_app.DefaultFilePath = @ubicacion
  excel_app.DisplayAlerts = false
  wkb = excel_app.workbooks.open xlpath
  wsh = wkb.sheets(xltable)
  rango = wsh.usedrange
  filas = Array.new
  rango.rows.each { |r|
    columnas = Array.new
    r.columns.each {|c|
      columnas.push c.value
    }
    filas.push columnas
  }
  rango = nil
  wsh = nil
  wkb.Close#(0)
  wkb = nil
  excel_app.Quit
  excel_app.ole_free
  excel_app = nil
  GC.start
  unless filas.length > 1 then
    puts "No hay datos!"
    abort
  end
  campos = filas.shift
  # creamos la tabla
  crea_tabla item_table, campos, filas
end
```

A4 – Salida.rb Caso 2

```
require 'C:/japath/sqlite-jdbc-3.8.10.1.jar'
require 'C:/sapjco/sapjco3.jar'
java_import com.sap.conn.jco.JCoTable
java_import com.sap.conn.jco.JCoDestinationManager
java_import com.sap.conn.jco.JCoContext

require 'nokogiri'
require 'jdbc/sqlite3'
require 'win32ole'
require 'sequel'

class Generada

  def proceso

    # Conexión a la base de datos
    @conn_work = java.sql.DriverManager.getConnection
    'jdbc:sqlite:work.sqlite'

    # inicializamos tablas y secuencia de procesos
    xltabin_materiales
    sapsql_materiales
    fictabout_materiales

    # Finalizams
    puts 'That''s all folks!!!'
    @conn_work.close
  end

private

  def crea_tabla item_table, campos, filas
    # creamos la tabla
    puts "Creamos la tabla:#{item_table}"
    @conn_work.create_table item_table.to_sym do
      campos.each do |c|
        send( 'COLUMN', c.to_sym )
      end
    end
    # insertamos los datos
    @conn_work[item_table.to_sym].import campos, filas
  end

  def xltabin_materiales

    xlp_path = "matparm.xlsx"
    xltable = "sapsql_materiales_parm"
    puts "Proceso xltabin; Path:#{xlp_path} Table:#{xltable}"
    begin
      excel_app = WIN32OLE.connect("excel.application")
    rescue
      excel_app = WIN32OLE.new("excel.application")
    end
    puts "Excel default file path:#{excel_app.DefaultFilePath}"
    excel_app.DefaultFilePath = @ubicacion
    excel_app.DisplayAlerts = false
    wkb = excel_app.workbooks.open xlp_path
    wsh = wkb.sheets(xltable)
    rango = wsh.usedrange
    filas = Array.new
```

```

rango.rows.each { |r|
  columnas = Array.new
  r.columns.each {|c|
    columnas.push c.value
  }
  filas.push columnas
}
rango = nil
wsh = nil
wkb.Close#(0)
wkb = nil
excel_app.Quit
excel_app.ole_free
excel_app = nil
GC.start
unless filas.length > 1 then
  puts "No hay datos!"
  abort
end
campos = filas.shift
# creamos la tabla
crea_tabla item_table, campos, filas

end
def sapsql_materiales

  item_code = "SELECT MI~MATNR_EXT MC~MATNR MT~MAKTX MC~WERKS
MC~DISPO\nMC~DISMM MC~BESKZ\nFROM MATERIALID AS MI \nINNER JOIN MAKT
AS MT ON MI~MATNR_INT = MT~MATNR\nINNER JOIN MARC AS MC ON
MI~MATNR_INT = MC~MATNR\nFOR ALL ENTRIES IN
sapsql_materiales_parm\nWHERE MT~SPRAS EQ 'S'\nAND MC~WERKS EQ
sapsql_materiales_parm-werks\nAND MC~DISPO EQ sapsql_materiales_parm-
dispo\nAND MC~DISMM EQ sapsql_materiales_parm-dismm"
  item_table = "sapsql_materiales"
  regex_groupby = /(.*)\bgroup by\b(.*)/im
  regex_limit = /(.*)\blimit\b(.*)/im
  regex_orderby = /(.*)\border by\b(.*)/im
  regex_tabla = /(.*)\forall entries in (\w+)/im
  regex_where = /where (.*)/im

  select = item_code.downcase
  tabla = ''
  antes_del_where = ''
  clausula_where = ''
  resto_clausulas = ''

  tmatch_tabla = select.match regex_tabla
  if tmatch_tabla then
    antes_del_where = tmatch_tabla[1]
    tabla = tmatch_tabla[2]
  else
    puts "Vaya! No encuentro tabla :-("
    abort
  end

  puts "Antes del where:#{antes_del_where}"
  puts "Tabla:#{tabla}"

  regex_campos = /\b(#{tabla})\-\S+\b/im

  tmatch_where = select.match regex_where

```

```

if tmatch_where then
    clausula_where = tmatch_where[1]
end

puts "Clausula where:#{clausula_where}"

tmatch_orderby = clausula_where.match regex_orderby
if tmatch_orderby then
    clausula_where = tmatch_orderby[1]
    resto_clausulas = " order by #{tmatch_orderby[2]}"
else
    tmatch_groupby = clausula_where.match regex_groupby
    if tmatch_groupby then
        clausula_where = tmatch_groupby[1]
        resto_clausulas = " group by
#{tmatch_groupby[2]}"
    else
        tmatch_limit = clausula_where.match regex_limit
        if tmatch_limit then
            clausula_where = tmatch_limit[1]
            resto_clausulas = " limit
#{tmatch_limit[2]}"
        end
    end
end

puts "Resto clausulas:#{resto_clausulas}"

campos_where = clausula_where.scan regex_campos

# creamos un dataset de la tabla de 'for all entries'
items = @conn_work[tabla.to_sym]

witems = Array.new
items.each do |i|
    witem = "( #{clausula_where.chomp} )"
    campos_where.each do |c|
        campo = c.first
        n = campo.gsub("#{tabla}-", '')
        witem.gsub!(campo, "#{i[n.to_sym]}")
    end
    witems.push witem
end

item_code = "#{antes_del_where} where #{witems.join("\nor
")} #{resto_clausulas}".upcase
puts item_code

# empezamos con la parte SAP

campos = Array.new
filas = Array.new
$ald = 'ALD'
destination = JCoDestinationManager.getDestination($ald)
JCoContext.begin destination
function = destination.getRepository.getFunction("Z_RFC_SQL")
tsqltab = function.getTableParameterList().getTable("TSQLTAB")
item_code.each_line do |l|
    tsqltab.appendRow()
    tsqltab.setValue('ZLINEA', l.chomp)
end

```

```

begin
  function.execute(destination)
rescue => mensaje
  puts "Error: #{mensaje.to_s}"
  return
end
txmldata = function.getTableParameterList().getTable("TXMLDATA")
s_xml = ''
txmldata.firstRow()
begin
  xmldata = txmldata.getValue('ZLINEA')
  s_xml << xmldata
end while txmldata.nextRow()
s_xml = s_xml[s_xml.index('<asx:abap')+1, s_xml.length-
s_xml.index('<asx:abap')+1]
doc = Nokogiri::XML(s_xml)
doc.xpath("//item").each_with_index {|rec, index|
  fila = Array.new
  rec.children.each { |campo|
    if index == 1 then
      campos.push campo.name.downcase
    end
    fila.push campo.content
  }
  filas.push fila
}
# creamos la tabla
crea_tabla item_table, campos, filas
end

```

```
def fictabout_materiales
```

```

  fl_path = 'materiales.txt'
  item_table = 'sapsql_materiales'
  # puts fl_path
  if File.file?(fl_path) then
    puts "Fichero #{fl_path} existe. Borrado!"
    File.delete(fl_path)
  end
  fl = File.new("#{fl_path}", 'w')
  puts "Fichero #{fl.path} creado!"
  ds = @conn_work.from(item_table.to_sym)
  fl.puts "#{ds.columns.join("\t")}\r"
  fl.puts "#{ds.map(ds.columns).collect{|a|
a.join("\t")}.join("\r\n")}\r"
  fl.close

```

```
end
```

```
end
```

```

proc = Generada.new()
begin
  proc.proceso
rescue => problem
  print "#{problem}\n"
  abort
end

```


Referencias

- [1] "About SAP SE/Company Information" [En línea]. Available: <http://www.sap.com/corporate/en/company.html>
- [2] "Forbes – The world biggest companies" [En línea]. Available: <http://www.forbes.com/global2000/>
- [3] "Panorama Consulting Services - 2015 ERP REPORT" [En línea]. Available: <http://go.panorama-consulting.com/rs/panoramaconsulting/images/2015%20ERP%20Report.pdf>
- [4] "SAP Help Portal" [En línea]. Available: <https://help.sap.com/>
- [5] "Wikipedia: Sistema de planificación de recursos empresariales" [En línea]. Available: https://es.wikipedia.org/wiki/Sistema_de_planificaci%C3%B3n_de_recursos_empresariales
- [6] Juan Larrañeta, Luís Onieva y Sebastián Lozano, "Métodos modernos de gestión de la producción", Alianza Editorial, S.A. Madrid 1988
- [7] "Wikipedia: ABAP" [En línea]. Available: <https://es.wikipedia.org/wiki/ABAP>
- [8] "abap open sql limitations" [En línea]. Available: <http://it.toolbox.com/blogs/sap-on-db2/abap-open-sql-limitations-31495>
- [9] Jack Herrington, "Code Generation in Action", Manning, Greenwich 2003
- [10] Paolo Perrotta, "Metaprogramming Ruby", The Pragmatic Bookshelf 2010
- [11] "ruby-doc.org" [En línea]. Available: <https://ruby-doc.org/>
- [12] "jruby.org" [En línea]. Available: <http://jruby.org/>
- [13] "jruby.org/apidocs" [En línea]. Available: <http://jruby.org/apidocs/>
- [14] "jruby wiki" [En línea]. Available: <https://github.com/jruby/jruby/wiki>
- [15] Charles O Nutter, Nick Sieger, Thomas Enebo, Ola Bini, Ian Dees, "Practical Using Jruby. Bringing Ruby to Java", The Pragmatic Bookshelf 2011
- [16] "sqlite.org" [En línea]. Available: <https://sqlite.org/>
- [17] "Sequel: The Database Toolkit for Ruby" [En línea]. Available: <http://sequel.jeremyevans.net/>
- [[18] "SAP Java Connector" [En línea]. Available: http://help.sap.com/saphelp_nwpi711/helpdata/en/48/70792c872c1b5ae1000000a42189c/content.htm
- [19] "zSQL: A tool to execute SQL statements directly in sap" [En línea]. Available: <https://blogs.sap.com/2013/10/20/zsql-a-tool-to-execute-sql-statements-directly-in-sap-source-code/>

Siglas

4GL	4th Generation Language
ABAP	Advanced Business Application Programming
API	Application Programming Interface
BAPI	Business Application Programming Interface
BI	Business Intelligence
BOM	Bill Of Materials
CRM	Customer Relationship Management
ERB	Embedded RuBy
ERP	Enterprise Resource Planning
GUI	Graphi User Interface
Jco	SAP Java Connector
JDBC	Java Database Connectivity
MRP	Material Requirement Planning
MRPII	Manufacturing Resource Planning
OLE	Object Linking and Embedding
ORM	Object Relational Mapping
RFC	Remote Funcion Call
SaaS	Software as a Service
SAP	Systeme, Anwendungen und Produkte in der Datenverarbeitung
SAP SE	SAP Societas Europaea
SOA	Service Oriented Architecture
SQL	Structured Query Language
SRM	Supplier Relationship Management
XML	eXtensible Markup Language