FORMALIZACIÓN DE UN PROCESO SOFTWARE INTEGRAL APLICANDO SOCCA

Silvia Teresita Acuña y Mabel del Valle Sosa
Universidad Nacional de Santiago del Estero, Avenida Belgrano (S) 1912, (4200) Santiago del Estero, Argentina.
E-mail: {silvac, litasosa}@unse.edu.ar

RESUMEN
La Ingeniería del Software (IS) y la Ingeniería del Conocimiento (IC) desarrollan sistemas de software mediante modelos de procesos de construcción diferentes. En este artículo se describe un Modelo de Proceso Software Integral (MPSI), aplicable en la construcción de sistemas convencionales (SC) y de sistemas basados en conocimientos (SBC). El modelo diseñado es prescriptivo, es decir indica "qué se hace" para producir y mantener una solución automatizada a un problema del mundo real. El modelo es formalizado mediante el enfoque de modelización orientado a objetos SOCCA (Especificación de las Actividades Coordinadas y Cooperativas). El modelo formal obtenido, que representa las tres P: los procesos, los productos y las personas, integra las perspectivas de datos, de proceso y de comportamiento en una representación comprensible para los usuarios del proceso software.

Palabras clave: Ingeniería de Software, enfoque SOCCA, proceso de formalización, orientación a objetos, proceso software integral.

1. INTRODUCCIÓN
El proceso software es un factor crítico en la entrega de sistemas de software de calidad [13], ya que tiene por finalidad gestionar, transformar y soportar la necesidad del usuario en un producto de software que satisface esa necesidad. El objetivo general de la investigación del proceso software es mejorar la práctica del desarrollo de software a través de [11]: a) formas mejoradas de diseño organizacional al nivel de procesos individuales y de la organización como un todo; y b) innovaciones complementarias en el soporte tecnológico. En este contexto, el proceso software se define como el conjunto de actividades necesarias para producir un sistema de software, ejecutadas por un conjunto de recursos humanos organizados según una estructura organizativa concreta y contando con un soporte de herramientas técnico-conceptuales.

Hoy por hoy, un área que juega un papel central en la investigación del proceso software es la modelización y definición del proceso [20, 5, 8, 15]. El objetivo último de la modelización del proceso software es orientar o guiar, controlar y gestionar las actividades del proceso. La prescripción del proceso de construcción de software es un tema estudiado en la IS desde hace algunos años. En 1991, la IEEE publicó un modelo cualitativo, informal y prescriptivo [9]. Desde entonces muchas otras propuestas han surgido [14, 6, 19, 12, 4, 8, 7, 10, 18, 16] procurando formalizar y automatizar el proceso de construcción.

En 1996, se ha desarrollado un Modelo de Proceso Software Integral (MPSI) aplicable en IS y en IC [1], validado y refinado en 1999 [2, 3]. La originalidad de este modelo es especialmente para la IC, donde no existe un proceso software definido. Aunque esta propuesta le resulte familiar a la IS, se plantean diferencias radicales con los modelos existentes: se da mayor importancia a las primeras fases de desarrollo, agregando nuevas actividades a estas fases. El MPSI no presenta una prescripción dinámica del proceso software que permita una comprensión más adecuada del mismo y que establezca una comunicación efectiva entre los subprocesos del proceso software.

El objetivo de este artículo es describir el Modelo de Proceso Software Integral y detallar su formalización mediante el método SOCCA que integra las perspectivas de datos (enfoque estático), de proceso y de comportamiento (enfoques dinámicos).

2. MODELO DE PROCESO SOFTWARE INTEGRAL

Denominamos Modelo de Proceso Software Integral al modelo de proceso software que prescribe la construcción de SC y de SBC y de software que integre SC y SBC. Es decir, es el modelo de proceso software que determina el "qué se hace" y el "quién lo hace", útil a la IS y a la IC.

El MPSI representado en la figura 1 [3] está compuesto de un subproceso que permite la selección de un modelo de ciclo de vida del software (MCVS) que se constituye en su eje y de otros tres subprocesos: uno que lo gestiona, otro que lo modeliza, y un tercero que asiste a la modelización. Estos subprocesos se han denominado respectivamente: Proceso del Modelo de Ciclo de Vida del Software, Procesos de Gestión del Proyecto, Procesos de Modelización del Software y Procesos Integrales de Soporte del Proyecto.

Dentro de los Procesos de Modelización del Software, los Procesos de Exploración, de Conceptualización, de Formalización y de Utilización se corresponden con los modelos exploratorios, conceptuales, formales y de utilización, respectivamente. Así, se crean los modelos exploratorios para la formulación del problema y la determinación de su viabilidad; los modelos conceptuales para la definición del sistema, la especificación de requisitos y la determinación de conocimientos; los modelos formales para la representación del diseño y de la implementación; y los modelos de utilización para la instalación, uso, retiro y mantenimiento del software.

A los procesos que asisten a los Procesos de Modelización del Software y son necesarios para completar con éxito las actividades del proyecto de software, se los denomina Procesos Integrales de Soporte del Proyecto. Integrales porque se aplican durante todo el ciclo de vida del software y a todos los aspectos que cada uno involucra y de Soporte porque apoyan a la realización de un producto de software fiable (Proceso de Verificación y Validación, Proceso de Configuración); que sea utilizado al máximo de sus capacidades (Proceso de Formación, Proceso de Documentación); y que ayuden a la constante interacción con resolutores (expertos), especialistas y usuarios (Proceso de Conformación del Equipo, Proceso de Adquisición de Información y Conocimientos). Los Procesos Integrales de Soporte del Proyecto son imprescindibles para asistir en su totalidad a la construcción de un software de calidad. Éstos son activados por los Procesos de Gestión del Proyecto y por los Procesos de Modelización del Software y por ellos mismos.

Se han agrupado, a su vez, en Procesos de Protección de la Calidad que incluyen los Procesos de Verificación y Validación y de Configuración; en Procesos de Transferencia Tecnológica que incluyen los Procesos de Documentación y de Formación; y, por último, en Procesos de Comunicación que incluyen los Procesos de Conformación del Equipo y de Adquisición de Información y Conocimientos.

Por razones de espacio, se describen sólo los elementos del Proceso de Diseño (sombreado en la figura 1), correspondiente a los Procesos de Formalización del MPSI. Estos elementos se formalizan mediante el método SOCCA en las secciones siguientes. Se selecciona el Proceso de Diseño porque constituye el proceso central que unifica la construcción y el mantenimiento del software. En la tabla 1 se presentan las actividades que se llevan a cabo en el diseño para transformar los documentos de entrada en documentos de salida.

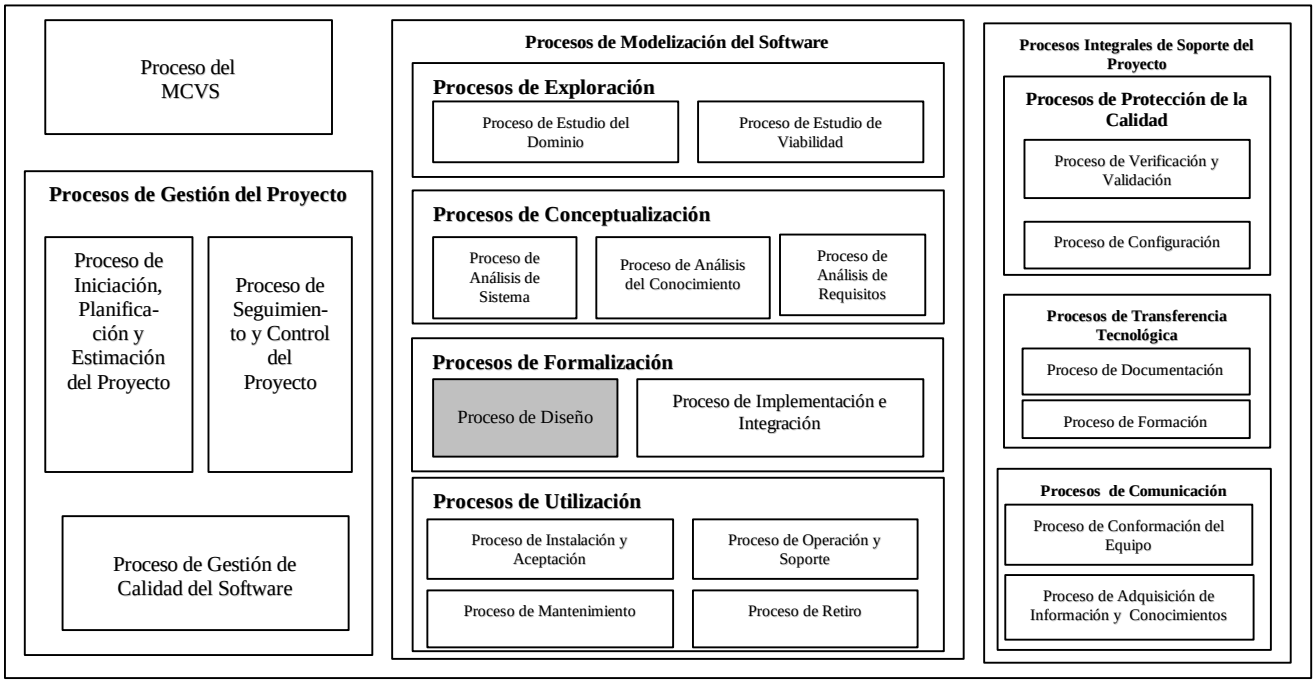

Figura 1: Modelo de Proceso Software Integral

Descripción	Documentos de Entrada	Actividades	Documentos de Salida
El Proceso de Diseño traduce el "qué hacer" de las especificaciones de los requisitos en el "cómo hacerlo" de las especificaciones de diseño. Inicialmente, la representación describe una visión sistémica y holística del software. Posteriores refinamientos de diseño conducen a una representación que se acerca al código fuente.	- Especificación de Requisitos del Software - Requisitos de Interfaz de Software - Plan de Gestión del Proyecto Software - Arquitectura del Sistema - Modelo del Conocimiento	1) Realizar el diseño preliminar 2) Analizar el flujo de información 3) Diseñar la base de datos (si se aplica) 4) Diseñar las interfaces 5) Seleccionar o Desarrollar: algoritmos y, o, formalismos del conocimiento 6) Realizar el diseño detallado	- Descripción de: Diseño Preliminar del Software y de Diseño del Software - Descripción de: Flujo de Información, Base de Datos, Interfaces y Algoritmos - Representación del Conocimiento del Resolutor

Tabla 1: Descripción, actividades, documentos de entrada y de salida del Proceso de Diseño del MPSI

3. FORMALIZACIÓN DEL MPSI

Para la formalización del MPSI se utiliza el método SOCCA [7], Especificación de las Actividades Coordinadas y Cooperativas, en inglés, Specifications of Coordinated and Cooperative Activities. Este formalismo, desarrollado en la Universidad de Leiden, permite la especificación del proceso software a partir de diferentes perspectivas: la de datos, la de proceso y la de comportamiento, y en donde cada una puede ser modelizada separadamente y al mismo tiempo como componentes integrados de una especificación completa. Tal especificación modular disminuye la complejidad de la actividad de modelización e incrementa la posibilidad de cambio y evolución de los modelos de proceso software.

Para modelizar la perspectiva de los datos se utilizan los diagramas de clase orientados a objetos, basados en conceptos del modelo entidad-relación extendido (ERE). La perspectiva de comportamiento y su coordinación se modeliza a través de diagramas de transición de estados (DTE) y PARADIGM. PARADIGM es un formalismo de especificación que fue originalmente desarrollado para y restringido a la especificación de procesos paralelos [17]. Para modelizar la perspectiva de proceso se emplean los diagramas de flujo de objetos (DFO). Los DFO son una versión extendida de los diagramas de flujos de datos con operaciones derivadas de las especificaciones de los diagramas de clase.

El enfoque de modelización de SOCCA considera los formalismos indicados en la figura 2 de la siguiente manera. A partir de la descripción de un problema, se construye un diagrama de clases, que tiene en cuenta tanto los atributos como las operaciones, llamadas operaciones de exportación de la clase. Sobre la base de estas operaciones de exportación, se desarrollan los DTE y PARADIGM. El comportamiento externo de cada clase es modelizado como un DTE, exhibiendo todas las secuencias posibles para responder los llamados de sus operaciones de exportación. Luego, a partir del estudio de dónde son importadas las operaciones de exportación, desde los diferentes comportamientos internos las operaciones de una clase son modelizadas como DTE, exhibiendo todas las secuencias posibles para convocar a las operaciones importadas. En el siguiente paso por reuso de los DTE que modelizan los comportamientos externos, denominados administradores de procesos, se coordina la comunicación entre los diferentes comportamientos internos, denominados empleados de procesos. Este enfoque se denomina PARADIGM [7]. Finalmente se describen las transformaciones de objetos y cómo los objetos fluyen de una transformación a otra por medio de los DFO.

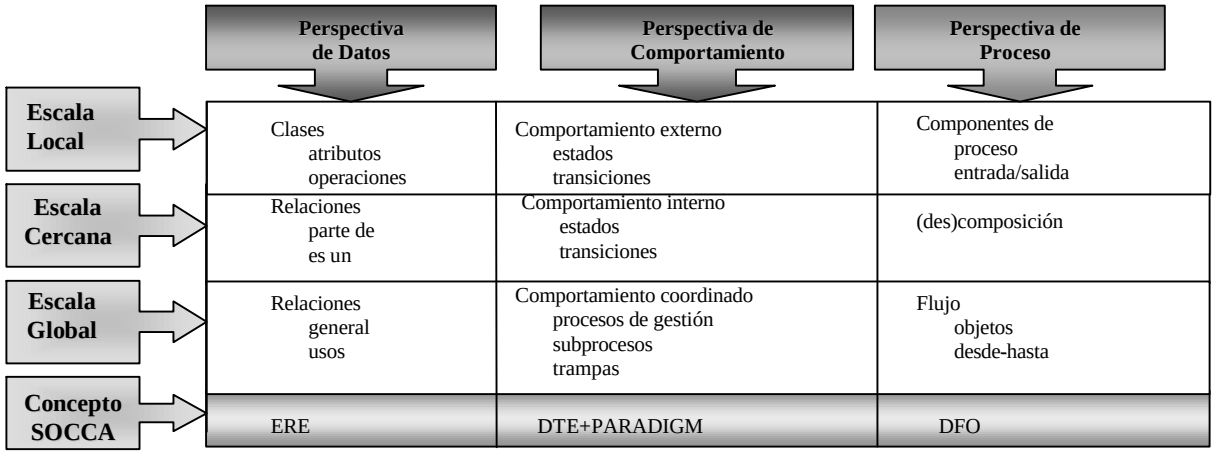

Figura 2: Perspectivas de especificación de diferente escala

Los pasos y subpasos de SOCCA para modelizar el proceso software se describen en la tabla 2.

	Pasos y Subpasos
PERSPECTIVA DEL PROBLEMA	*1. Describir el problema mediante una especificación textual o un diagrama de flujo de datos general y global.*
PERSPECTIVA DE DATOS	*2. Construir los diagramas de clase.* *2.1. Describir los objetos estructurados complejos mediante jerarquía de clases.* *2.2. Adicionar las relaciones generales faltantes.* *2.3. Determinar cuáles operaciones usan operaciones exportadas desde otras clases.*
PERSPECTIVA DE COMPORTAMIENTO	*3. Construir los diagramas de transición de estados.* *3.1. Especificar el comportamiento externo para cada clase.* *3.2. Indicar qué operaciones se importan desde alguna parte para cada clase.* *3.3. Especificar el comportamiento interno de cada operación para cada clase.* *4. Aplicar PARADIGM* *4.1. Describir el comportamiento secuencial de cada proceso mediante la utilización de DTE.* *4.2. Identificar los subdiagramas significativos o subprocesos teniendo en cuenta la coordinación con otros procesos dentro de cada DTE.* *4.3. Identificar los estados "trampas", dentro de cada subproceso.* *4.4. Describir las transiciones y comportamientos entre los subprocesos de los objetos.*
PERSPECTIVA DE PROCESO	*5. Construir los diagramas de flujo de objetos.*

Tabla 2: Proceso de modelización. Pasos y subpasos del método SOCCA

4. APLICACIÓN DE SOCCA AL PROCESO DE DISEÑO

A partir de la especificación textual de la tabla 1 para el Proceso de Diseño del MPSI, los pasos que se realizan para la modelización son los que se describen a continuación:

Paso 1- Diseño del diagrama de clases

En este paso se definen los objetos como jerarquías de clase y las relaciones de "parte de" y "es un". Además se añaden las relaciones generales y se indican los métodos que utilizarán las operaciones exportadas desde otra clase. El diagrama de clases, los atributos y operaciones y las relaciones generales se representan en las figuras 3, 4 y 5 respectivamente. Como último subpaso, se define un nuevo tipo de relación binaria, llamada *usos*, que indica las instancias donde las operaciones de exportación serán importadas. Los usos definidos se indican en la figura 6.

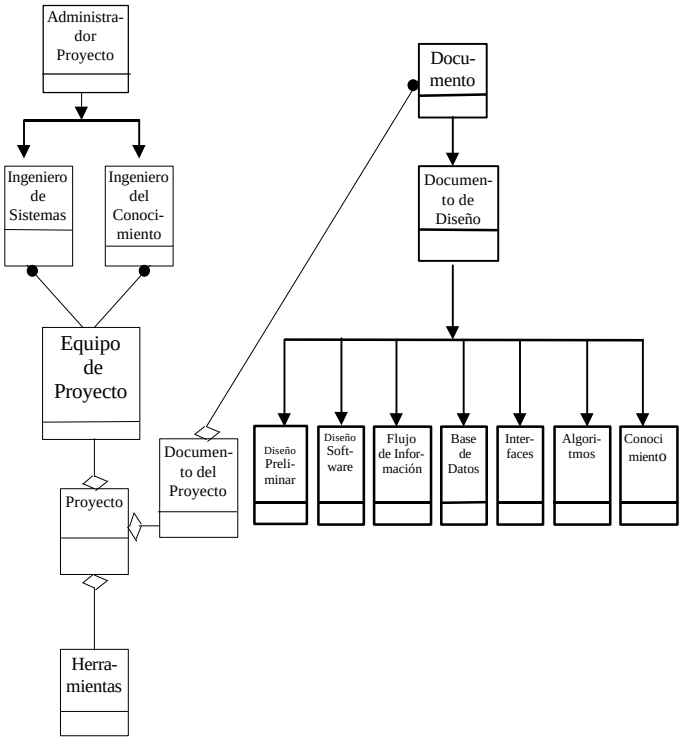

Figura 3: Diagrama de clases

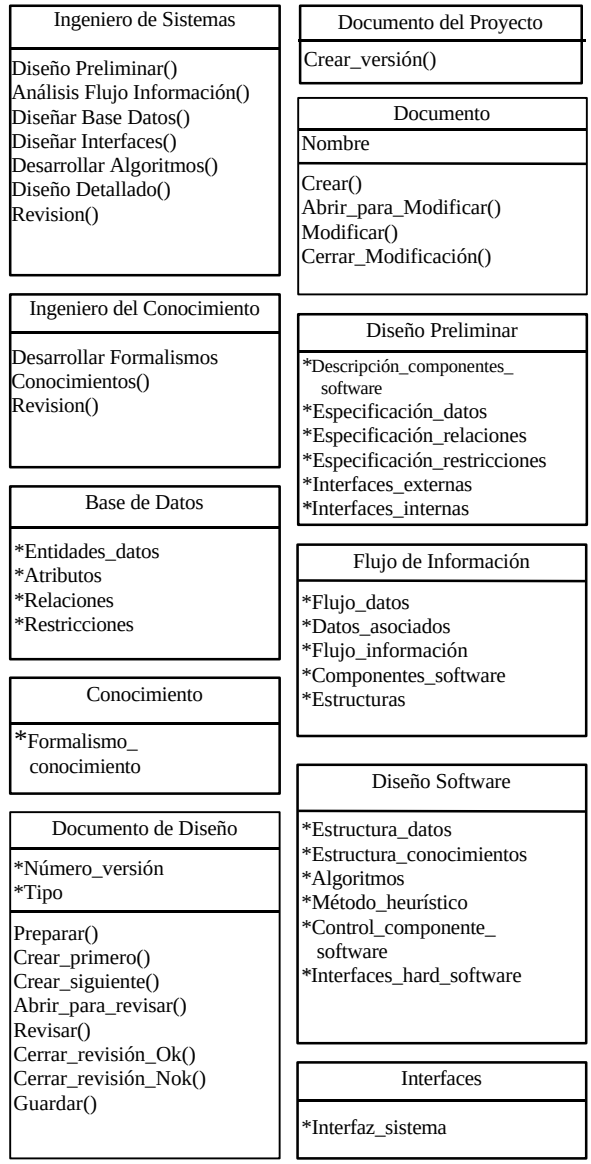

Figura 4: Diagrama de clases: atributos y operaciones

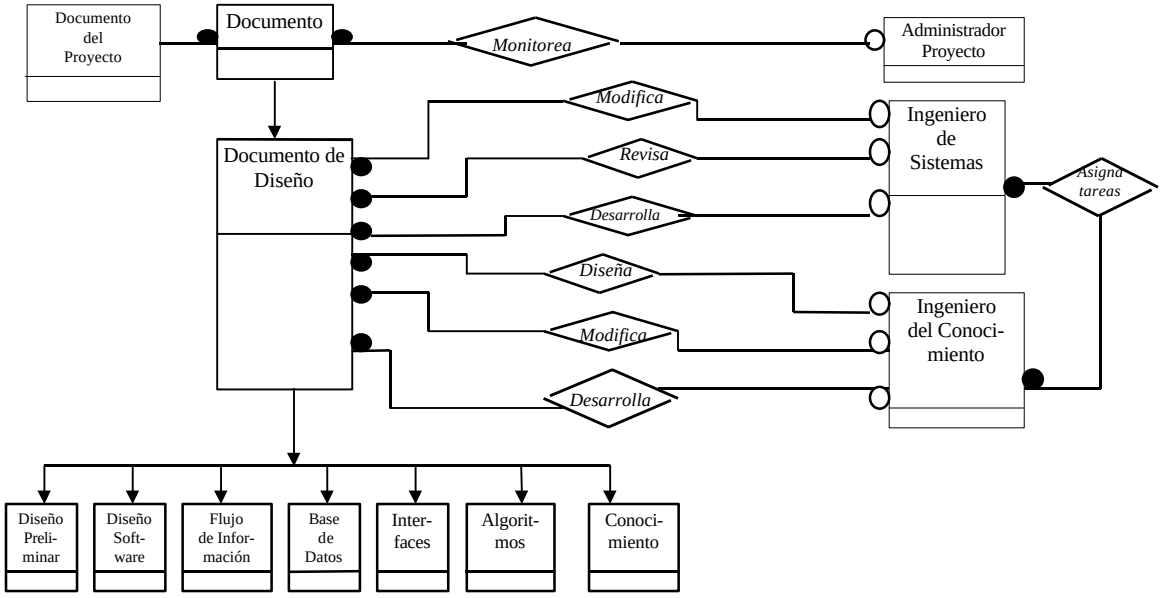

Figura 5: Diagrama de clases: clases y relaciones generales

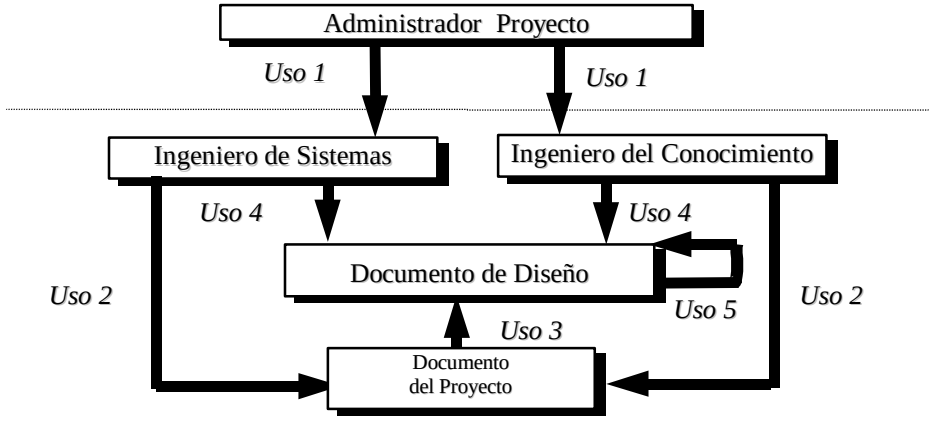

Figura 6: Diagrama de Importación-Exportación de Operaciones

Paso 2- Diseño de diagramas de transición de estados

En este paso se define el comportamiento de una clase. Para tal fin se llevan a cabo los siguientes subpasos:

a) Para cada clase se especifica el comportamiento externo, que se define como el comportamiento visto desde afuera. Este comportamiento especifica la secuencia de llamadas de las operaciones de exportación.
b) Se especifican las operaciones (exportadas) importadas desde alguna parte.
c) Se especifica para cada clase, el comportamiento interno de cada operación. Este comportamiento consiste principalmente de las secuencias de llamadas con respecto a las operaciones importadas.

A partir de los STD se especifica el comportamiento externo del *Ingeniero de Sistemas* y luego el comportamiento externo del *Documento del Proyecto* correspondientes al "Diseño Preliminar". Las operaciones se ordenan en una lista de operaciones de exportación del *Ingeniero de Sistemas*, teniendo en cuenta aquellas que sean relevantes para producir una nueva versión de *Documento* (figura 7).

En la figura 8 se presenta el STD del comportamiento externo del *Ingeniero de Sistemas*. En este diagrama, todos los estados son rotulados con un comentario breve que indica la interpretación de un estado y las transiciones se identifican con una operación exportada. El estado rotulado como Neutral es el estado al que el *Ingeniero de Sistemas* regresa después de cada actividad que ha sido iniciada por él. Los estados rotulados como Inicio de Diseño Preliminar, Inicio de Análisis de Flujo de Información, Inicio de Diseño de Base de Datos, etc. indican el comienzo de cada actividad.

Diseño Preliminar(Nombre.Doc)
Análisis Flujo Información(Nombre.Doc)
Diseño Base Datos(Nombre.Doc)
Diseño Interfaces(Nombre.Doc)
Desarrollar Algoritmos(Nombre.Doc)
Desarrollar Formalismos Conocimientos(Nombre.Doc)
Diseño Detallado(Nombre.Doc)
Revision()

Figura 7: Lista de operaciones de exportación del Ingeniero de Sistemas

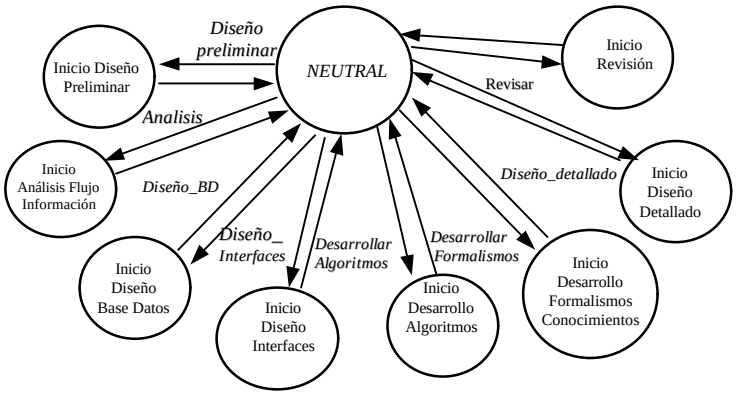

Figura 8: STD del comportamiento externo del Ingeniero de Sistemas

Teniendo en cuenta la clase *Documento del Proyecto*, la operación de exportación es: *Crear_Versión* (Nombre.Doc). Como las versiones de otros documentos pueden ser creadas después que una creación ya ha comenzado pero antes que haya terminado, el comportamiento externo sólo especifica el inicio de la creación, seguido por el retorno tan rápido como sea posible al estado Neutral. Esto se representa en la figura 9.

La clase final cuyo comportamiento externo se modeliza es *Documento de Diseño*. Previamente se elabora una lista de operaciones para describir su comportamiento. La lista completa de operaciones se muestra en la figura 10.

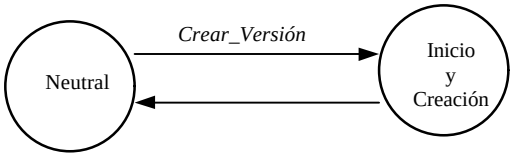

Figura 9: STD del comportamiento externo de Documento del Proyecto

Preparar
Crear_primero
Crear_siguiente
Abrir_para_modificar
Modificar
Cerrar_modificación
Abrir_para_revisar
Revisar
Cerrar_revisión_Ok
Cerrar_revisión_Nok
Guardar

Figura 10: Operaciones de comportamiento externo de Documento de Diseño

Considerando las órdenes posibles de estas operaciones debe tenerse en cuenta que cada instancia de *Documento de Diseño* representa exactamente una versión de un *Documento de Diseño*. Para esta versión donde el comportamiento es secuencial se propone el diagrama de la figura 11.

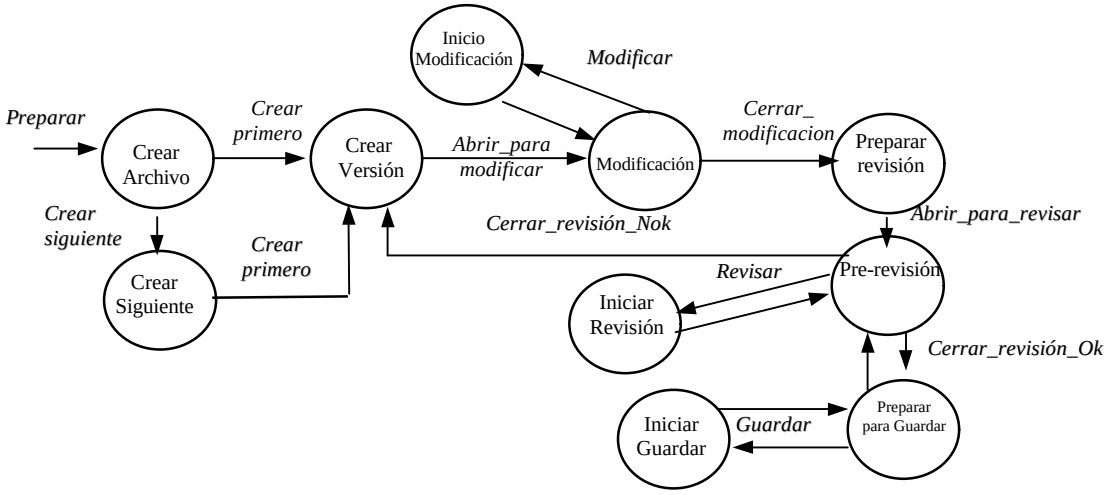

Figura 11: STD del comportamiento externo de Documento de Diseño

Una vez definidos los comportamientos externos, se especifican los comportamientos internos de las operaciones. Para ello, se presenta cada relación *usos* con una lista posible de operaciones importadas (figura 12).

Uso 1
Diseño Preliminar
Análisis Flujo Información
Diseño Base Datos
Diseñar interfaces
Desarrollar Algoritmos
Diseñar Formalismos
Conocimientos
Diseño Detallado

Uso 2
Crear Versión

Uso 3
Crear primero
Crear siguiente

Uso 5
Guardar
Preparar

Uso 4
Abrir_para_modificar
Modificar
Cerrar_modificación
Abrir_para_revisar
Revisar
Cerrar_revisión_Ok
Cerrar_revisión_Nok
Guardar

Figura 12: Definición de usos

Dependiendo del comportamiento de un determinado objeto, solamente un subconjunto de estas operaciones es realmente importada. Por ejemplo un *Ingeniero del Conocimiento* necesita en un determinado momento sólo la operación *Diseñar Formalismos de Conocimientos*.

A continuación, se especifica el comportamiento interno de las operaciones. Para nombrar estos comportamientos internos se antepone el prefijo *int* al nombre de las operaciones exportadas.

Las operaciones que se consideran para especificar el comportamiento interno son: Int_Diseño_ preliminar, Int_Revisión, Int_Crear_Versión, Int_Crear_Siguiente, Int_Ok. A modo de ejemplo se representan los dos primeros en las figuras 13 y 14.

Dentro de la especificación del comportamiento interno, se definen tres nuevos tipos de operaciones:

a) Las operaciones que tienen antepuesto el prefijo *call*, indican la/s operación/es exportada/s desde alguna parte.
b) Las operaciones que no tienen *call*, indican que son operaciones internas.
c) Las operaciones que tienen el prefijo *act*, reflejan alguna comunicación entre el comportamiento interno y el externo.

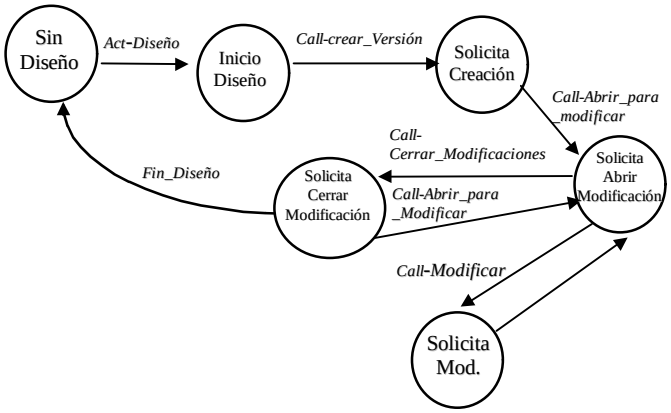

Figura 13: STD del comportamiento interno de Int-Diseño_preliminar

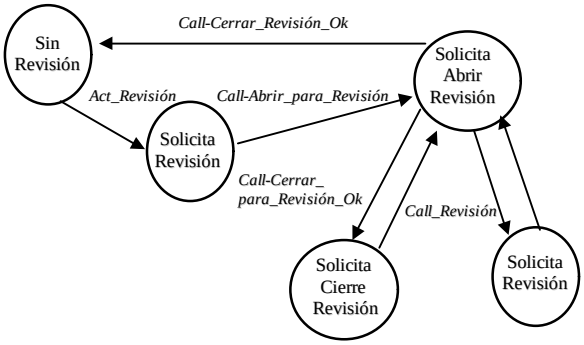

Figura 14: STD del comportamiento interno de Int_Revisión

Paso 3- Aplicación de PARADIGM

Los distintos tipos de comportamiento mostrados en el paso anterior son estrictamente secuenciales. Es decir, la ejecución de cualquier operación de exportación que ocurre en algún lugar de comportamiento externo inicia la ejecución del correspondiente comportamiento interno y, por el contrario, convocar una operación de exportación desde dentro del comportamiento interno inicia la ejecución de esa operación de exportación. Para la aplicación PARADIGM se deben determinar dos categorías de comunicación existente entre dos comportamientos secuenciales. Estas categorías se denominan comunicación tipo 1 y comunicación tipo 2.

Tipo 1

En esta categoría se encuadra la comunicación que se da entre el comportamiento interno y algún comportamiento externo de otro objeto.

En la figura 15 se representa el momento en que se estable la comunicación, como resultado inmediato de convocar desde un comportamiento interno una operación mientras ésta es exportada desde otro objeto.

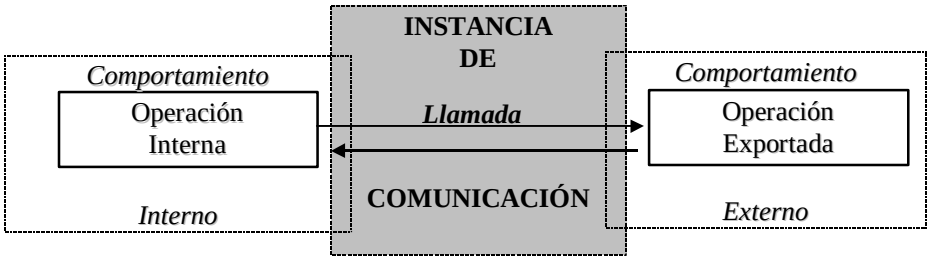

Figura 15: Representación gráfica de la comunicación de tipo 1

Tipo 2

Esta categoría está dada por la comunicación entre cualquier comportamiento externo y el comportamiento interno que pertenece al mismo objeto.

En la modelización de PARADIGM se realizan los siguientes pasos:

1. Descripción del comportamiento secuencial mediante la utilización de STD.
2. Identificación de los subdiagramas significativos o subprocesos, teniendo en cuenta la coordinación con otros procesos.
3. Identificación de los estados “trampas”, dentro de cada subproceso.
4. Descripción de las transiciones y comportamientos entre los subprocesos de los objetos.

Para la descripción de relaciones de tipo 1, se toma el comportamiento externo del *Ingeniero de Sistemas* como administrador. Los empleados son los comportamientos internos del *Ingeniero de Sistemas*: *Int_Diseño_preliminar, Int_Análisis_flujo_información, Int_Diseñar_Base_Datos, Int_Diseñar interfaces, Int_Desarrollar_algoritmos, Int_Diseñar_formalismos_conocimientos, Int_Diseño_detallado e Int_ Revisión.* En la figura 16 se representan los subprocesos y trampas de Int_Diseño_preliminar e Int_Revisión.

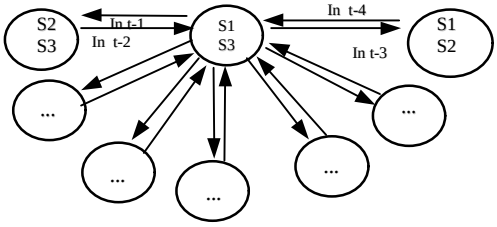

Figura 16**:** Ingeniero de Sistemas como administrador de Int_Diseño_preliminar e Int_Revisión

En las figuras 17 y 18 se detallan los subprocesos y trampas de Int_Diseño_preliminar y de Int_Revisión. Por ejemplo, supongamos el llamado a *Diseño Preliminar* desde alguna parte, parametrizado con un documento llamado Nombre.Doc; como resultado de esta operación el *Ingeniero de Sistemas* transita desde Neutral al Inicio Diseño, parametrizado con Nombre.Doc. Y como el *Ingeniero de Sistemas* está fuera de Inicio Diseño, significa que Int_Diseño_preliminar está en Subproceso 1 (S1) y en trampa t-1. Solamente después que Int_Diseño_preliminar ha alcanzado la trampa t-1, el *Ingeniero de Sistemas* puede transitar de Neutral a Inicio Diseño, con el parámetro Nombre.Doc.
Cuando el *Ingeniero de Sistema*s ingresa en Inicio Diseño, prescribe los subprocesos S2 a Int_Diseño_ preliminar. Como Int_Diseño_preliminar estuvo en trampa t-1, inmediatamente comienza dentro del subproceso 2 (S2), desde el estado Sin Diseño, por esa razón entra casi inmediatamente a la trampa t-2. Al ingresar a t-2 el *Ingeniero de Sistemas* abandona Inicio Diseño y transita a Neutral, de esta forma prescribe de nuevo el subproceso S1 a Int_Diseño_preliminar, dentro del cual puede hacer las llamadas necesarias y al final ingresar a la trampa t-1.
Mientras no se ingrese a t-1, el mismo *Ingeniero de Sistemas* puede comenzar cualquier revisión o cualquier otro diseño, luego de haber sido convocado para esa actividad.
La coordinación entre varios Ingenieros de Sistemas que trabajan con un mismo Documento Nombre.Doc es controlada por el comportamiento externo de este documento único.

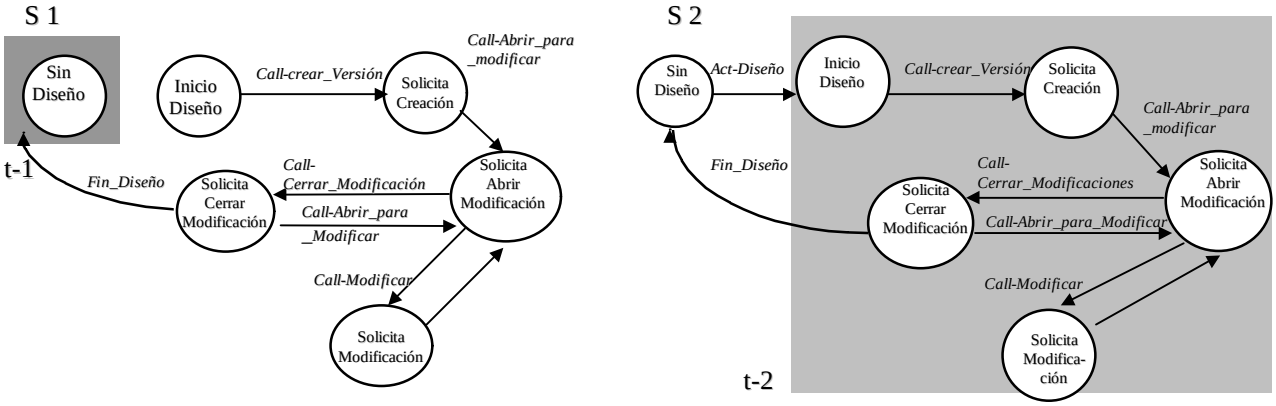

Figura 17: Subprocesos y trampas de Int_Diseño_preliminar con respecto al Ingeniero de Sistemas

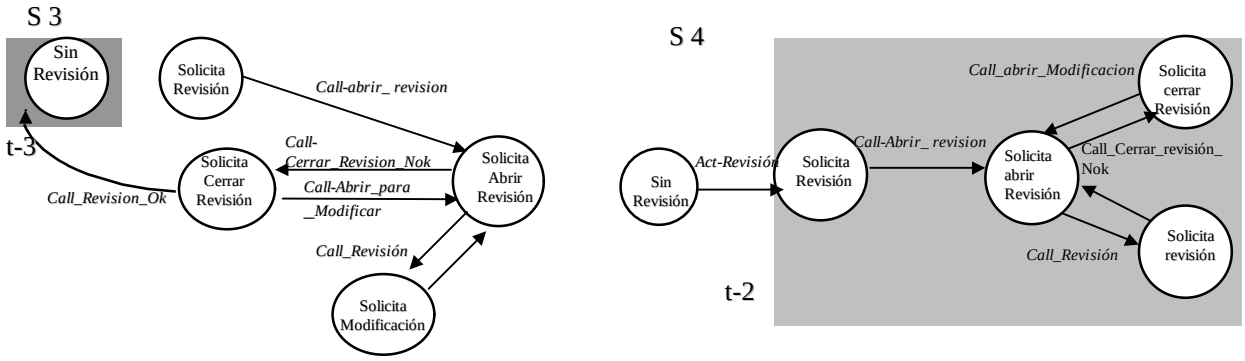

Figura 18: Subprocesos y trampas de Int_Revisión con respecto al Ingeniero de Sistemas

En cuanto a la comunicación de tipo 2, se enfatiza el control de dependencias, orden y prioridades entre los comportamientos que pertenecen a un objeto. Por ejemplo para el objeto *Ingeniero de Sistemas*, se remarcan dos comportamientos internos y uno externo, los que constituyen tres roles. Para realizar estos tres roles adecuadamente es necesario alternar de uno a otro rol, punto en el cual se produce la comunicación de tipo 2.

5. CONCLUSIONES

El Modelo de Proceso Software Integral formalizado es prescriptivo, estático y dinámico para guiar en el desarrollo de sistemas de software convencional y basado en conocimientos, de modo separado o conjunto. Este modelo integra los elementos procesos, actividades, productos y personas y sus interacciones asociadas del proceso software. En este artículo se describe la modelización de las actividades, de los documentos de entrada y de salida, de las personas y sus roles para el Proceso de Diseño del MPSI. Esta modelización, a través de SOCCA, se realiza a partir de diferentes perspectivas: la de datos, la de proceso y la de comportamiento. Los formalismos tradicionales para describir el proceso software se centran en una única perspectiva, sin embargo el enfoque de SOCCA combina a un alto nivel de abstracción lo mejor de diferentes formalismos: modelización orientada a objetos basada en la modelización de entidad-relación extendida, diagramas de transición de estados en combinación con PARADIGM y diagramas de flujo de objetos. El modelo formal obtenido asegura el recubrimiento y modelización de todos los elementos influyentes del proceso software mediante un método adecuado e integral.

Además, utilizando SOCCA se formaliza el MPSI, teniendo en cuenta tanto las partes técnicas del proceso software, como también las partes humanas, o mejor dicho los miembros del equipo de trabajo del modelo propuesto. Normalmente, las partes humanas están ausentes en los modelos de procesos software tradicionales.

Así, con el modelo de proceso software integral formalizado se logra que los profesionales dedicados al desarrollo de SC y SBC dispongan de una herramienta técnico-conceptual y formal orientada a objetos que los asista en la producción de software de calidad.

6. REFERENCIAS

[1] S. T. Acuña y N. Juristo, “Software process model for KBS development”. The Journal for the Integrated Study of Artificial Intelligence, Cognitive Science and Applied Epistemology. *CC-AI* **13**, 4 (1996) 1-39.

[2] S. T. Acuña y C. Lasserre, *Proceso Software: Un Modelo para la Ingeniería del Software y del Conocimiento*. (EDUNJU, 1999).

[3] S. T. Acuña, J. Ares, N. Juristo, J. L. Morant y A. Pazos, "A process model applicable to software engineering and knowledge engineering". *International Journal of Software Engineering and Knowledge Engineering* (1999) 1-30. En prensa.

[4] S. C. Bandinelli, A. Fuggetta y C. Ghezzi, "Software process model evolution in the SPADE environment". *IEEE Trans. Software Engineering* **19**, 12 (Diciembre 1993) 1128-1144.

[5] B. Curtis, M. Kellner y J. Over, "Process modeling". *Communications of the ACM* **35**, 9 (Septiembre 1992) 75-90.

[6] W. Deiters y V. Gruhn, "Software process analysis based on FUNSOFT nets". *Systems Analysis Modelling Simulation* **8**, 4-5 (1991) 315-325.

[7] G. Engels y L. Groenewegen, *"SOCCA: Specifications of coordinated and cooperative activities"*. In Software Process Modelling and Technology cap. 4. (Research Studies Press, 1994) 71-102.

[8] J. Finkelstein, Kramer y B. Nuseibeh, *Software Process Modelling and Technology.* (Research Studies Press, 1994).

[9] *IEEE Standard for Developing Software Life Cycle Processes*, IEEE Standard 1074-1991.

[10] ISO/IEC *International Standard: Information Technology. Software Life Cycle Processes*, ISO/IEC Standard 12207-1995.

[11] P. Kawalek y D. G. Wastell, "Organisational design for software development: a cybernetic perspective". In *Lecture Notes in Computer Science, Software Process Technology: Proceedings of the 5th European Workshop* 1149 (Springer-Verlag, 1996) 258-270.

[12] M. I. Kellner, "Software process modelling support for management planning and process modelling". *Proceedings of the First International Conference Software Process* (Octubre 1991) 8-28.

[13] M. M. Lehman, "Software engineering, the software process and their support", *Software Engineering Journal* **6**, 5 (Setiembre 1991) 243-258.

[14] J. Lonchamp, K. Benali, C. Godart y J. C. Derniame, "Modeling and enacting software processes: an analysis". *Proceedings of the 14th Annual International Computer Software and Applications Conference* (Octubre-Noviembre 1990) 727-736.

[15] I. R. McChesney, "Toward a classification scheme for software process modelling approaches". *Information and Software Technology* **37**, 7 (1995) 363-374.

[16] Sang-Yoon Min y Doo-Hwan Bae, "MAM nets: A Petri-net based approach to software process modeling, analysis and management". *Proceedings of the 9th International Conference on Software Engineering and Knowledge Engineering* (Junio 1997) 78-86.

[17] P. J. A. Morssink, "Behaviour Modelling in Information Systems Design: Application of the PARADIGM formalism". Ph.D. Tesis. Universidad de Leiden (1993).

[18] M. N. Nguyen y R. Conradi, "Towards a rigoroues approach for managing process evolution". In *Lecture Notes in Computer Science, Software Process Technology: Proceedings of the 5th European Workshop* 1149. Springer-Verlag (1996) 18-35.

[19] M. H. Penedo y C. Shu. "Acquiring experiences with the modelling and implementation of the project life-cycle process: the PMDB work". *Software Engineering Journal* **6**, 5 (Septiembre 1991) 259-274.

[20] C. Tully, "Representing and enacting the software process". *Proceedings of the 4th International Software Process Workshop*, ACM SIGSOFT Software Engineering Notes **14**, 4, (Junio 1989).