MODELIZACIÓN Y VERIFICACIÓN AUTOMÁTICA DE UN PROCESO SOFTWARE

Mabel del V. Sosa, María I. Ledesma[1], **Mariela Lescano**[1]
Departamento de Informática
Facultad de Ciencias Exactas y Tecnologías
Universidad Nacional de Santiago del Estero
Avenida Belgrano (S) 1912, (4200)
Santiago del Estero, Argentina.
litasosa@unse.edu.ar, {lesdesma, lescam}@arnet.com.ar

Silvia T. Acuña
Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Avenida Tomás y Valiente 11
28049 Madrid, España
silvia.acunna@uam.es

Resumen

En este artículo se presenta la modelización y verificación de los aspectos dinámicos y temporales de un proceso software convencional. La modelización de los aspectos dinámicos se representa mediante un metamodelo de diagramas de actividades de UML y la modelización de los aspectos temporales con un modelo de Red de Petri. Posteriormente se comprueban las propiedades funcionales y temporales con el lenguaje de lógica temporal CTL (Computational Tree Logic). Tanto para la modelización como para la verificación se utiliza AToM^3, una herramienta que posibilita el metamodelado mediante la utilización de multiformalismos, y la verificación automática mediante técnicas de comprobación o model checking.

En este trabajo ha sido posible analizar el comportamiento del proceso para determinar la coherencia y consistencia de las especificaciones efectuadas como resultado de los requerimientos del diseño, permitiendo detectar los posibles fallos del modelo y efectuar las correcciones correspondientes. Así mismo se destaca la importancia de contar con herramientas de metamodelación, multiformalismos y verificación integrados, como soportes fundamentales de apoyo a los métodos convencionales de análisis y diseño del proceso software.

Palabras clave: Modelo de proceso software, modelización dinámica temporal, verificación formal, lenguaje de lógica temporal, model checking.

1. Introducción

El proceso software es un factor crítico en la entrega de sistemas de software de calidad, ya que tiene por finalidad gestionar, transformar y soportar la necesidad del usuario en un producto de software que satisface esa necesidad [1]. El objetivo general de la investigación del proceso software es mejorar la práctica del desarrollo de software a través de: 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.

Debido a que el objetivo último de la modelización del proceso software es guiar, controlar y gestionar las actividades del proceso, los aspectos dinámicos del proceso software deben ser considerados de modo preciso. Esta modelación del proceso software requiere de métodos formales de especificación que permitan el estudio de sus propiedades y, en particular, determinen con precisión sobre la corrección de sus desarrollos [17][22]. Existen diversas propuestas [10][15][16][18] procurando formalizar y automatizar el proceso de construcción. Sin embargo, se observa una carencia de especificaciones formales y temporales de la dinámica del proceso software [4][23] que permita una comprensión más adecuada del mismo y que establezca una comunicación efectiva entre los subprocesos del proceso software.

---

[1] Tesistas de grado del Proyecto "Gestión Integrada de las Organizaciones: Modelización de las Capacidades Humanas en los Procesos de Software", Código 23/C051, aprobado por el CICyT-UNSE, Argentina.

Para realizar estas especificaciones, existen lenguajes temporales [5][14][19][20] que combinan la expresividad provista por la lógica con la posibilidad de realizar referencias a nociones de orden temporal provista por operadores específicos agregados a un cálculo proposicional. No obstante, se requiere un formalismo capaz de modelar y representar de forma explícita y gráfica la concurrencia o paralelismo de las actividades, y determinar el tiempo para completar las actividades o la totalidad de las mismas, y predecir el orden temporal en que pueden ocurrir los eventos. Es decir, representar a través de un lenguaje preciso, común y claro a todos los participantes los aspectos para modelar la dinámica del proceso.

El objetivo de este artículo es presentar la formalización de un Modelo de Proceso Software Integral (MPSI) desde la perspectiva dinámica temporal y la verificación del modelo resultante. Para modelar los aspectos dinámicos, división de trabajo y responsabilidades, se utiliza UML siguiendo las especificaciones de [24], además para completar la formalización del modelo se incluirá el aspecto temporal con Redes de Petri, considerando secuencias, concurrencia o paralelismo de actividades y orden temporal en que pueden ocurrir los eventos. Finalmente la verificación de las propiedades dinámicas temporales del modelo resultante se realiza mediante un lenguaje de lógica temporal CTL (Computational Tree Logic). Los procedimientos mencionados se realizan aplicando AToM^3 [7][9] que permite la modelización con distintos formalismos y la verificación del modelo respectivamente.

El artículo se organiza como sigue: en la sección 2 se describe el Modelo de Proceso Software Integral. En la sección 3 se presentan aspectos conceptuales de model checking como método de verificación formal y una descripción global de la herramienta AToM^3. En la sección 4 se detalla el procedimiento para la modelización del Subproceso de Iniciación, Planificación y Estimación del MPSI y la validación del modelo obtenido. En la sección 5 se presentan los trabajos relacionados y finalmente en la sección 6 se exponen las conclusiones del trabajo.

2. Modelo de Proceso Software Integral (MPSI)

Se ha desarrollado un Modelo de Proceso Software Integral [2] aplicable a la construcción de sistemas convencionales implementado en una herramienta que está disponible en http://www.ls.fi.upm.es/spt/. Este modelo de proceso software convencional representa las interacciones entre los elementos: actividades, productos, roles y técnicas involucradas en un desarrollo. El MPSI está compuesto de un subproceso que permite la selección de un modelo de ciclo de vida del software 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 de Selección 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. La estructura de MPSI está representado mediante diagramas de clases usando la notación de UML [24]. Se han definido los objetos como jerarquías de clase, los atributos, las operaciones y las relaciones generales. Una descripción completa del modelo se puede encontrar en [2].

En este trabajo se definen sólo los elementos del Subproceso Iniciación, Planificación y Estimación correspondientes al Proceso de Gestión del MPSI. Se seleccionan los subprocesos de gestión porque crean la estructura del proyecto y aseguran el nivel apropiado de la gestión del mismo durante todo el ciclo de vida del software, permitiendo, dadas las limitaciones en tiempo y dinero que existe habitualmente para la construcción de un producto de software, alcanzar este objetivo de un modo no sólo eficaz sino eficiente en el consumo de recursos, incluidos los humanos. Los aspectos estructurales de estos elementos están formalizados mediante diagramas de clases como se muestra en la figura 1.

En la figura 1 se definen dos grandes grupos de clases: Rol y Documento. Debido a razones de simplicidad sólo el nombre de las clases se consigna en esta figura. Las clases Equipo del Proyecto, Usuario y Proveedor del Sistema forman parte de otra clase, Proyecto. El Equipo del Proyecto está integrado por uno a muchos Miembros. Cada Miembro puede desempeñar uno o más roles tales como el de Planificador y otros (ingeniero de calidad, controlador) que no se muestran en la figura. Estas clases comparten todos los atributos de la clase Miembro. Cada Miembro utiliza una o más técnicas aplicables a determinados subprocesos de MPSI. A su vez, la clase Miembro forma parte de la clase Equipo del Proyecto. Esta representación permite una asignación dinámica, ya que las instancias representadas pueden cambiar en tiempo de ejecución. La clase abstracta Documento tiene hijos tales como: Plan de Soporte, Plan de Retiro,

Plan de Gestión del Proyecto (PGP) y Plan de Adquisición de Conocimientos (PAC). Estas clases heredan todas las operaciones de la clase padre, Documento. Cada clase tiene atributos y operaciones. En la figura 2 se presentan las principales clases del subproceso de Iniciación, Planificación y Estimación del proyecto.

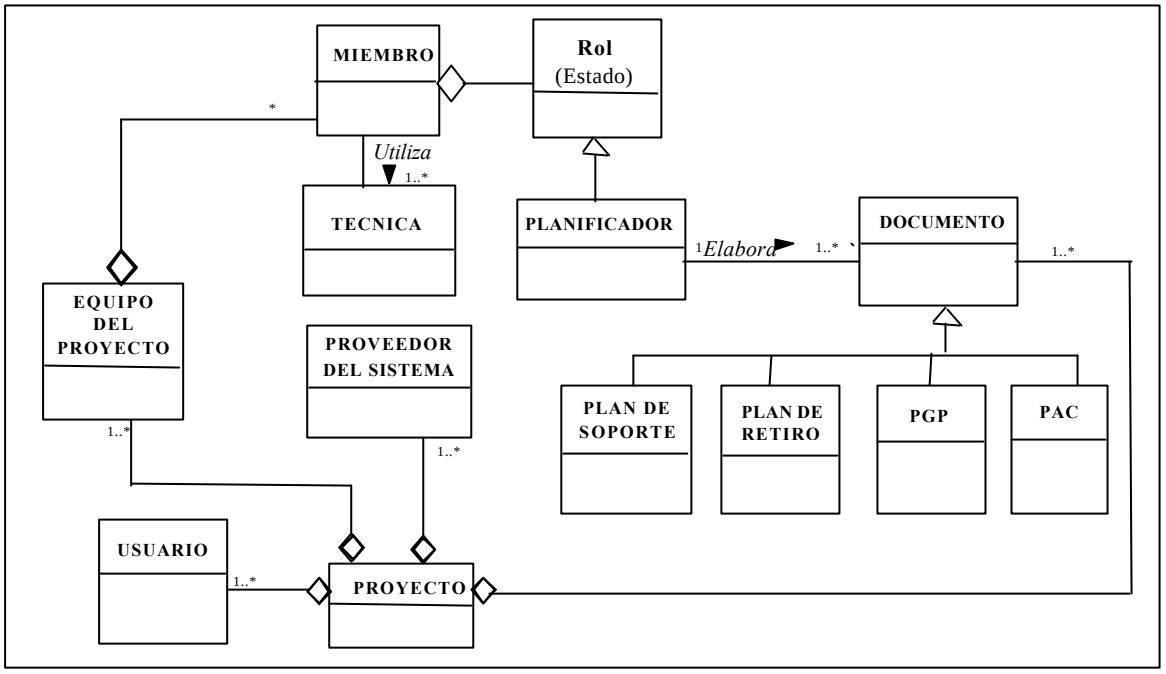

Figura 1. Diagrama de clases del Subproceso de Iniciación, Planificación y Estimación del proyecto

Figura 2. Principales clases del Subproceso de Iniciación, Planificación y Estimación del proyecto

3. Método de verificación formal: Model Checking

Una meta en Ingeniería de Software es permitirle a los desarrolladores de software construir sistemas que alcancen máxima fiabilidad a pesar de su complejidad. Una manera de lograr esta meta es usando métodos formales, los que ayudan a encontrar inconsistencias, ambigüedades e incompletitudes [6][11].

Los métodos formales son técnicas y herramientas basadas en lenguajes matemáticos que sirven para especificar sistemas y comprobar si estos cumplen con las propiedades deseadas o las especificaciones descritas. Una de las técnicas de verificación automática de sistemas de estados finitos se llama model checking. Esta técnica se basa en dos conceptos fundamentales, que son la lógica proposicional temporal y

los diagramas de transición de estados. La idea subyacente consiste en utilizar una lógica de tiempo ramificado para expresar las condiciones temporales que deben satisfacerse y luego un autómata temporizado para realizar una exploración de los posibles estados por los que el sistema podría atravesar. De este modo se verificará si las condiciones expresadas en la sentencia lógica son satisfechas por el comportamiento del sistema.

3.1. Lenguaje de lógica temporal (CTL)

El lenguaje de lógica temporal CTL (lógica de árboles de computación) fue propuesto por Emerson y Clarke [11] como un lenguaje de especificación para sistemas de estados finitos. La idea básica consiste en utilizar una lógica de tiempo ramificado para expresar condiciones temporales que deben satisfacerse y un autómata temporizado para comprobar que todos los estados alcanzables del sistema verifiquen dichas condiciones.
La sintaxis de la lógica CTL es construida usando los *operadores lógicos usuales*, *proposiciones*, *cuantificadores de camino* y *cuantificadores temporales*.
Los *cuantificadores de camino* son: **A** (*en todos los caminos*) y **E** (*existe un camino*).
Los *cuantificadores temporales* son: **F** (*en algún camino futuro*), **X** (*en un próximo estado*), **G** (*en todos los estados futuros*), **U** (*operador until*) y **V** (*operador release*).

Un modelo **M** para la lógica CTL es una tupla **(S, Þ , L, P)** donde:

- **S** es un conjunto de estados,
- **Þ Í S** *x* **S** es una relación binaria llamada relación de transición tal que **"** s **Ï S $** s´ **Ï S**: s **Þ** s´,
- **P** es un conjunto de proposiciones, y
- **L**: **S** 2^P es una función de rotulación de estados.

La semántica de CTL se define formalmente con respecto a una estructura de Kripke[2]. Sea **M** = **(S, Þ , L, P)** un modelo para la lógica CTL. Dado un estado **s Ï S**, se dice que una fórmula f es verdadera en el estado **s** si solamente en M□ f. La relación de satisfacción □ es definida por inducción estructural de la fórmula f.

3.2. AToM^3

AToM^3 (A Tool for Multi-Formalism Modelling and Meta-Modelling) es una herramienta creada por un grupo de investigadores de la Universidad Autónoma de Madrid, España [3][7]. La misma es seleccionada porque facilita la formalización asistida por computador de sistemas complejos tal como el proceso software mediante la aplicación de conceptos de *multiformalismo* y *metamodelización* permitiendo la creación, edición, optimización y transformación de metamodelos. En general se puede decir que AToM^3 tiene una capa de *metamodelización*, a partir de la cual se pueden generar diferentes modelos en distintos formalismos representándolos en forma gráfica. Por ejemplo desde la *metaespecificación* de un diagrama entidad-relación, AToM^3, puede generar modelos descriptos con algunos formalismos, tales como diagrama de clase, redes de Petri, entre otros [8][9].
En AToM^3 el metamodelado permite representar modelos mediante lenguajes visuales o formalismos gráficos que usan una notación de alto nivel denominado metaformalismo. Una vez generado el metamodelo, es posible transformarlo en un modelo equivalente expresado en otro formalismo siempre que las transformaciones entre formalismos hayan sido definidas. Estas transformaciones se especifican internamente con gramática de grafos.
En el ámbito de la verificación de sistemas AToM^3 proporciona una técnica de verificación automática denominada model checking, orientada a realizar la verificación mediante la exploración exhaustiva del espacio de estados del modelo. en busca de situaciones que no cumplan con las propiedades especificadas. En este último caso la prueba es parcial, pero conservativa, en el sentido de que si el sistema cumple las propiedades, entonces el sistema real también lo hace, en caso contrario no se podría obtener una conclusión.

---

[2] Una estructura de Kripke es una generalización de las estructuras de la lógica clásica, que proveen las bases de la semántica para esa lógica.

AToM^3, provee un método de especificación que permite expresar los requerimientos formalmente mediante CTL y un mecanismo para la incorporación de un conjunto de reglas que permiten determinar si el sistema satisface los requerimientos indicados.

En síntesis, usa la Lógica Temporal CTL para especificar y comprobar las propiedades del sistema, a partir de los modelos representados mediante Redes de Petri, contra especificaciones en CTL.

4. Aplicación de model checking al MPSI

Cada una de las partes de MPSI puede ser modelizado separadamente y al mismo tiempo como componentes integrados de una especificación completa. La 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 dinámica temporal se utilizan diagramas de actividades de UML y Redes de Petri. Seguidamente, la verificación se orienta a comprobar las propiedades de comportamiento, secuencialidad y concurrencia del modelo a distintos niveles de abstracción [13].

Para la aplicación del model checking se realizan los pasos que se detallan a continuación:

1. **Modelización** del sistema MPSI, en particular el subproceso de Iniciación, Planificación y Estimación, mediante un metamodelo diagrama de actividades de UML.
2. **Transformación** del metamodelo a un modelo de red de Petri mediante reglas de transformación de formalismos de AToM^3.
3. **Especificación** de las propiedades que debe satisfacer el sistema, usando un lenguaje de lógica temporal CTL.
4. **Verificación** de la satisfacción de las propiedades especificadas en el sistema modelado.

4.1. Modelización

Para la modelización de los aspectos dinámicos del subproceso Iniciación, Planificación y Estimación se usa diagramas de actividades de UML desarrollados en un nivel de abstracción global, que expresan la concurrencia de las acciones y la división de responsabilidades y roles, tal como se muestra en la figura 3.

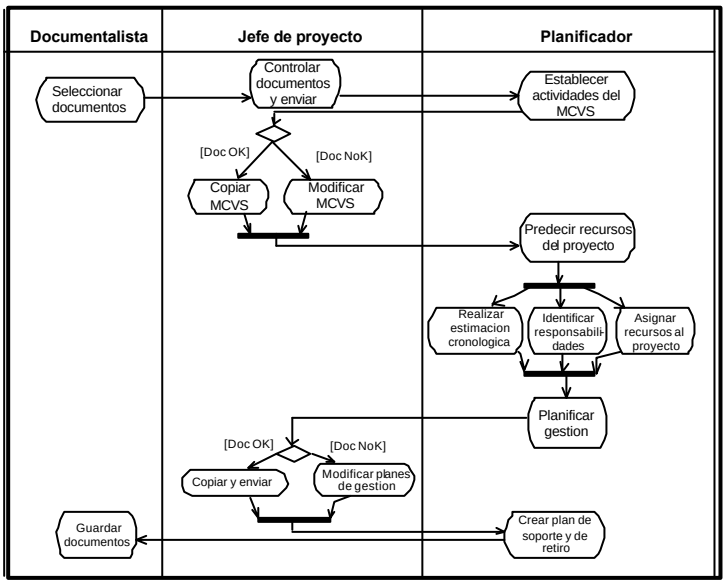

Figura 3. Diagrama de actividades del Subproceso de Iniciación, Planificación y Estimación

Se han utilizado estados de acción, para describir las actividades, por ejemplo "*Establecer actividades del MCVS*". Además se usan símbolos de condición, por ejemplo "*aceptar documento*" para habilitar las transiciones entre una acción y otra, un símbolo para los puntos de decisión representada por un diamante

con una o más transiciones de entrada y dos o más transiciones de salida etiquetadas con condiciones, por ejemplo "*documento Ok OR documento No Ok*". La línea gruesa horizontal dibujada en la transición "*enviar documento*" que sale de la acción "*Predecir recursos del proyecto*" es usada para indicar que la transición resulta en varias acciones paralelas, como son "*Realizar estimación cronológica*", "*Identificar responsabilidades*" y "*Asignación de recursos del proyecto*", las que pueden ejecutarse en cualquier orden o en forma paralela. Por último, se especifica la división de trabajo o responsabilidades, para lo que se utilizan carriles etiquetados, como por ejemplo "*Planificador*", "*Jefe de proyecto*" y "*Documentalista*" que representan las responsabilidades asignadas de acuerdo a los roles específico e intervinientes para realizar las acciones involucradas en el Proceso de Iniciación, Planificación y Estimación.

4.2. Transformación

A partir del diagrama de actividades se realiza la traducción a un modelo temporizado, considerándose la secuencia y el orden de cada actividad, tal como se muestra en la figura 4. El diagrama de red temporizado representa la evolución temporal cualitativa del sistema modelado que se orienta a representar las relaciones de orden definidas en el sistema. A partir de este modelo temporizado resultante es posible obtener información sobre la estructura y el comportamiento dinámico del sistema a diferentes niveles de abstracción. Concretamente, en este trabajo, se modeliza el orden, secuencia y concurrencia de las actividades involucradas en el Subproceso de Iniciación, Planificación y Estimación del MPSI.

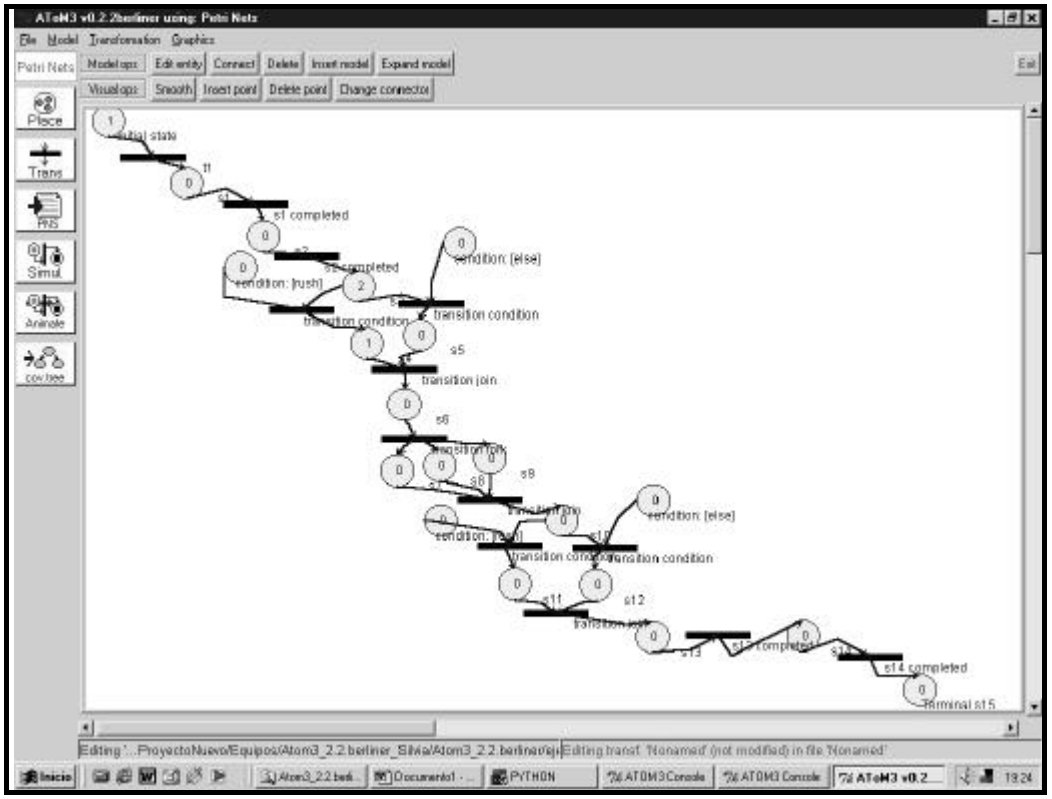

Figura 4. Modelo de Red de Petri del Subproceso de Iniciación, Planificación y Estimación

La Red de Petri se define como la tupla (S,T,F,m_0) donde: S es el conjunto de lugares, T es el conjunto de transiciones, $F \subseteq (SxT) \cup (TxS)$ es el conjunto de arcos y m_0 es el marcado inicial. En la red de la figura 3 cada lugar y transición están etiquetados por una letra mayúscula numerada que representa la actividad o acción que la provoca. En la tabla 1 se definen algunos lugares y transiciones de la red de Petri. A cada lugar o transición se le podría asociar características temporales cuantitativas pertinentes, que en este trabajo no serán consideradas.

Tabla 1. Definición de los componentes de la red de Petri

P: lugares	T: transiciones
S_1: Seleccionar documentos para el proyecto. S_2: Se controlan los documentos S_3: Establecer actividades acorde al MCVS. S_4: Controlar el mapa de actividades y MCVS. S_5: Modificar el mapa de actividades. S_6:Predecir recursos necesarios para ejecutar el proyecto. S_7: Realizar estimación cronológica del proyecto. S_8: Identificar las responsabilidades. S_9: Asignar recursos al proyecto. S_{10}: Planificar la gestión del proyecto (PGPS). S_{11}: Controlar el PGPS. S_{12}: Modificar PGPS. S_{13}: Crear plan de soporte y plan de retiro. S_{14}: Guardar documentos.	t_2: Enviar documentos. t_3: Enviar copia de los documentos. t_4: Enviar versión previa del documento MCVS. t_5: Enviar versión controlada del documento MCVS. t_6: Enviar versión final del documento mapa de actividades y MCVS. t_7: Iniciar actividad S_6. t_8: Enviar informe de recursos necesarios para el proyecto. t_9: Iniciar actividad S_7. t_{10}: Iniciar actividad S_8. t_{11}:Iniciar actividad S_9. t_{12}: Recibir versión final de documentos de S_7, S_8 y S_9 e iniciar actividad PGPS.

Con los pasos 1 y 2 se completan la modelización de los aspectos dinámicos y temporales del proceso de Iniciación, Planificación y Estimación del MPSI.

4.3. Especificación de propiedades dinámicas y temporales

Para la especificación de las propiedades temporales, relacionados a la concurrencia y secuencialidad, se considera el uso de la lógica temporal CTL para evaluar los posibles trayectos o caminos de evolución de un proceso. La evolución de un proceso puede interpretarse como una sucesión de estados por los que atraviesa la ejecución. Posteriormente se asocian a los distintos estados las propiedades que son válidas en dicho estado.

A veces es útil poder determinar: a)- Si cierta propiedad puede permanecer a través de todo el proceso de computación, b) Si alguna vez cierta propiedad se verifica, c)- Si existe intermitencia entre los momentos de validez de cierto par de propiedades relacionadas, d)- La persistencia de cierta propiedad.

Algunas de las principales propiedades son:

1. **Seguridad** (safety) **[+]A**: permite expresar *"alguna condición indeseada nunca sucede".*
2. **Terminación** (termination, guarantee, liveness). **<+> A**: permite expresar *"alguna condición deseada eventualmente sucede".*
3. **Persistencia** (persistence).**<+>[+] A**: En algún momento se llega a un estado en el que siempre se satisface **A**. Permite expresar que puede haber una demora antes que una propiedad se estabilice.
4. **Response**. Luego de que una condición **p** sucede, otra condición **q** se lleva a cabo.

Con respecto a la **evaluación de los caminos** se utilizan los operadores temporales (**A**, "*para todos los caminos*"; **E**, "*para algún camino*"; **EX**, "*para algún camino en el siguiente estado*"; **U**, "*hasta*"), los operadores booleanos (^ *conjunción*, **Þ** *implicación*, **Û** *equivalencia*) y los cuantificadores de camino (**"** , "*para todos los caminos"*, **$**, "*existe algún camino*").

Considerando el modelo formalizado del Subproceso de Iniciación, Planificación y Estimación, se han analizado diferentes cuestiones. A continuación se presentan algunos de los tipos de propiedades que se verifican con su correspondiente especificación en lógica temporal:

a. El proceso solo se bloquea al llegar a la actividad final guardar documentos. Si el proceso se inicia en la actividad S_1: *Seleccionar documentos para el proyecto*, el proceso sólo se bloquea al llegar a la actividad final S_{14}: *Guardar los documentos.*

$$\mathbf{"\,(}\ S_1\ \mathbf{U}\ S_{14}\mathbf{)}$$

b. Existe algún camino por el cual se llega al final correcto. Si se inicia el proceso en la actividad S_1, eventualmente existe un camino por el cual se alcanza el estado final S_{14}.

$$\mathbf{Ø\ \$X}\ S_1\ \mathbf{Ù\ Ø}\ S_{14}\mathbf{)}$$

c. Existe la posibilidad de que nunca se realice la actividad de modificación. Una vez completado el *control del mapa del ciclo de vida del software,* si no se detectan inconsistencias es posible que nunca se alcance el próximo estado S_5: *modificación de documentos*.

$$\emptyset \$ X\ S_5$$

Con respecto a la concurrencia y secuencialidad se verifican las siguientes propiedades:

d. Una vez recibido los documentos de entrada: "*Especificación de requisitos"*, "*Informe de necesidades y recursos"*, *"Modelo de ciclo de vida de software",* se establecen las actividades correspondientes al modelo de ciclo de vida del software adoptado, es decir, *se elabora el mapa de actividades*.

$$\emptyset AX\ f_1 \Rightarrow \emptyset AX\ S_3 \models_1 = \text{"Especificación de requisitos"} \wedge \text{"Informe de necesidades y recursos"} \wedge \text{"Modelo de ciclo de vida de software"}$$

e. Una vez producido el mapa de actividades, *se controla y se verifica de que no existan inconsistencias*.

$$AX\ S_3 U\ S_4$$

f. Si se detectan errores en el mapa de actividades establecido, se procede a *realizar las modificaciones*. Siempre que se detectan errores en la actividad S_3: *control de mapa de actividades* el próximo estado que eventualmente se debe alcanzar es S_5: *modificación de documentos*.

$$\emptyset AX\ S_3 \Rightarrow \emptyset\ AX\ S_5$$

g. La estimación cronológica para el proyecto se realiza después de predecir los recursos necesarios. Esto junto con la identificación de responsabilidades y asignación de recursos son prerrequisitos para planificar la gestión del proyecto.

$$AXf_1\ U\ S_{10} \models_1 = S_6 \wedge S_8 \wedge S_3$$

4.4. Verificación

Seguidamente, la verificación se orienta a comprobar las propiedades de comportamiento, secuencialidad y concurrencia del modelo a distintos niveles de abstracción [13].

La verificación del modelo desarrollado tiene básicamente los siguientes objetivos:

- Asegurar que el sistema no quede bloqueado en algún estado o actividad, es decir, que cada vez que se inicie una actividad, la misma finalice.
- Verificar aspectos de correctitud y seguridad, es decir que se alcancen estados permitidos y que no se llegue a estados indebidos.
- Determinar el orden de precedencia, concurrencia y secuencialidad de las actividades.

La verificación de las propiedades de vivacidad (aseguramiento de que no haya deadlock) y alcanzabilidad (aseguramiento que se puedan alcanzar todos los estados) se ha realizado mediante el trazado del "árbol de alcanzabilidad", en el cual se representa el espacio total de estados de la red, el gráfico completo tiene 55 estados y parte del mismo es presentado en la figura 5. Se puede observar que existen algunos *interleaving* debido a la posible ejecución en paralelo o simultaneidad de algunas actividades. Los resultados obtenidos han sido satisfactorios para ambas propiedades.

En la figura 6 se presenta un ejemplo de comprobación de la especificación realizada con AToM^3. En la figura se consigna la formula $\forall(\phi_1\ U\ \phi_2)$, donde ϕ_1 = actividad inicial S_1 y ϕ_2 = actividad final S_{14}. El resultado que devuelve la herramienta es la secuencia de estados por donde transita el proceso hasta alcanzar el estado final "*guardar los documentos*" donde concluye el mismo.

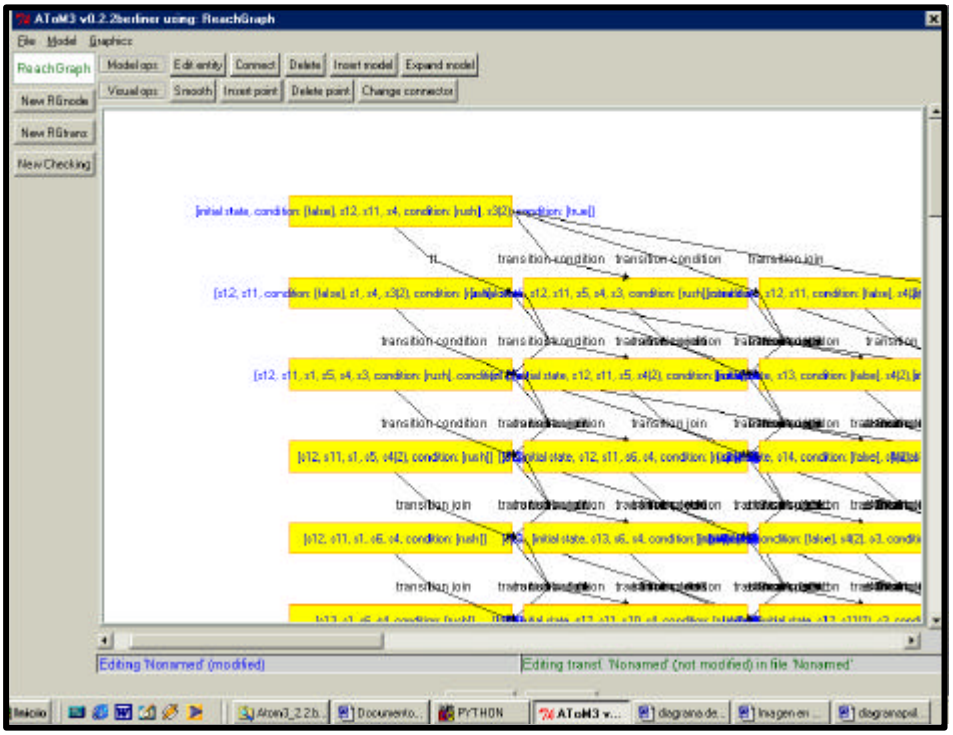

Figura 5. Árbol de alcanzabilidad del Subproceso de Iniciación, Planificación y Estimación

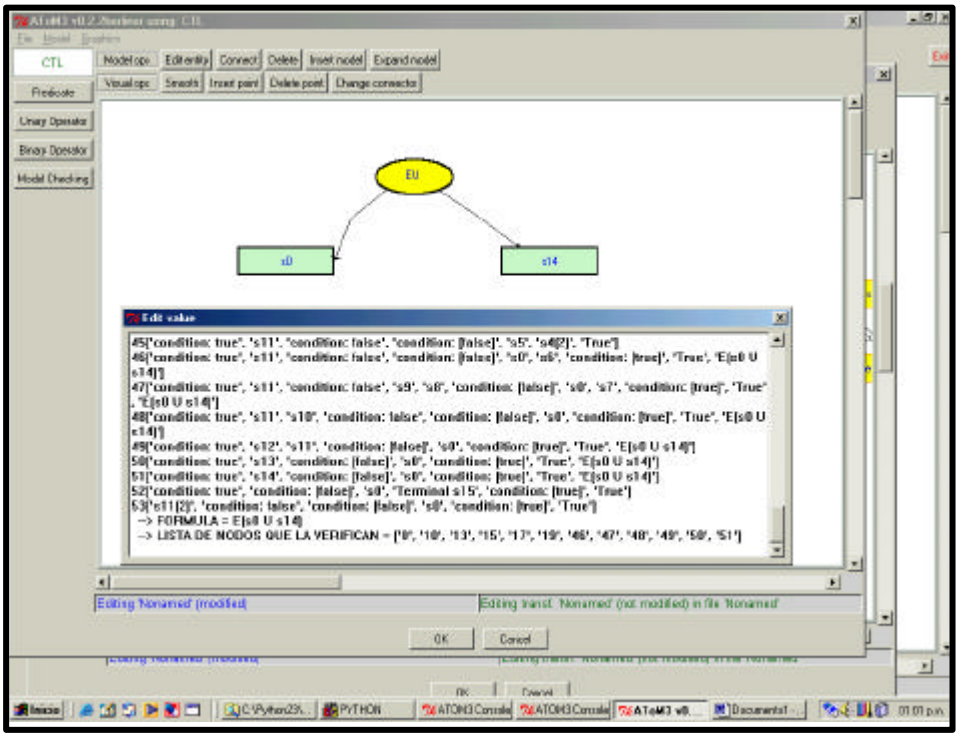

Figura 6. Interfaz gráfica para la comprobación de especificaciones CTL a través de model checking de AToM^3

Con los modelos presentados se logra clarificar y precisar el orden, tanto precedencia como simultaneidad de las actividades llevadas a cabo dentro del Proceso de Iniciación, Planificación y Estimación, a un nivel de detalle general, determinando los puntos de control, decisión y responsabilidades. Sin embargo en esta formalización se pretende considerar cuestiones más sutiles, de duración cuantitativa explícita para la simulación de características específicas del modelo. Por ejemplo determinar cuanto dura una actividad o cuanto tiempo debe o puede mediar entre dos actividades cualesquiera del proceso.

Para realizar este tipo de especificación cuantitativa se utiliza una extensión natural agregando subíndices a los operadores temporales para limitar su ámbito. Por ejemplo se puede escribir **$<5 p** para decir que **p** será verdadera a lo largo de algún camino de computación, pero dentro de las próximas **5** unidades de tiempo. Esas características son consideradas por TCTL (Timed Computation Tree Logic) todavía no soportadas por AToM^3.

Lo expuesto anteriormente constituye la base para verificar en el modelo presentado propiedades tales como:

- "Existe la posibilidad de que el proceso demore mas de **n** unidades de tiempo",
- "Existe la posibilidad de terminar antes de **n** unidades de tiempo",
- "Una vez que se inicia la actividad *Planificar gestión del proyecto,* el documento con el plan estará terminado a lo sumo en **n** días luego de iniciada la actividad".

Los aspectos mencionados serán considerados en un trabajo futuro.

5. Trabajos Relacionados

Existen diferentes tipos de modelizaciones del proceso. Los procesos se pueden modelar a diferentes niveles de abstracción (por ejemplo, modelos genéricos versus modelos personalizados, modelos descriptivos versus modelos prescriptivos versus modelos proscriptivos). La diferenciación entre estos tipos de modelos ha sido descrita en [21]. La estructuración de los tipos de información en un modelo de proceso puede realizarse a partir de diferentes perspectivas. Se presenta la siguiente lista de *perspectivas de la información* comunes encontradas en la literatura:

- **Funcional**: representa cuáles elementos del proceso se están realizando, y cuáles flujos de entidades de información son relevantes a esos elementos del proceso.
- **Comportamiento**: representa cuándo (por ejemplo, secuencialidad) y bajo qué condiciones los elementos del proceso se realizan.
- **Organizacional**: representa dónde y por quién en la organización, los elementos del proceso se realizan.
- **Informacional**: representa las entidades informacionales producidas o manipuladas por un proceso, incluyendo su estructura y relaciones.

Los modelos de proceso se construyen según los lenguajes o formalismos creados para representar la información específica sobre estas características. El lenguaje más usado en la práctica es el lenguaje natural (estructurado) debido a su flexibilidad. La tabla 2 presenta una lista de abstracciones de representación (excluyendo el lenguaje natural) organizadas según los puntos de vista mencionados previamente. Ninguna de estas abstracciones cubren todas las clases de información. Por esta razón, la mayoría de los modelos encontrados en la literatura presentan lenguajes basados en más de una de estas abstracciones. Estos modelos consideran la necesidad de la integración de múltiples paradigmas de representación (es decir, diferentes modelos de proceso), aunque esto generalmente determina una mayor complejidad en la definición del modelo [22].

Por una parte, como se puede observar en la tabla 2, hay una gran variedad de notaciones que han sido usadas para modelar los procesos. Por otra parte, una variedad de enfoques multi-paradigmas [12] han sido propuestos para modelar los procesos software centrados alrededor de una paradigma principal. Hay dos características principales de estos enfoques, en primer lugar, con excepción principalmente de los enfoques basados en redes de Petri temporales, todos ellos usan representaciones textuales y no temporales. En segundo lugar, la mayoría de ellos ya está en un nivel de abstracción de lenguajes de programación. Así, dificultan la modelación directa de una situación problemática, mientras que fuerzan al diseñador a tomar en cuenta muchos aspectos técnicos del formalismo de modelación [22]. Estas dos carencias han sido cubiertas en la formalización dinámica del proceso software aquí presentada. Además, de lograr cubrir las cuatro perspectivas consignadas en la modelación del proceso software a través de los formalismos orientados a objetos y temporales utilizados en este artículo (filas sombreadas en tabla 2).

Tabla 2. Bases de los lenguajes aplicables en las perspectivas de información del proceso

Base del Lenguaje	Perspectivas de la Información
Lenguaje de programación procedimental	Funcional – Comportamiento Informacional
Análisis y diseño de sistemas, incluyendo diagrama de flujo de datos y técnica de análisis y diseño estructurado (SADT)	Funcional – Organizacional - Informacional
Lenguajes y enfoques de inteligencia artificial, incluyendo reglas y pre-/ post-condiciones	Funcional - Comportamiento
Eventos y disparadores // Flujo de control	Comportamiento
Transición de estados, redes de Petri temporales // Statecharts	Funcional – Comportamiento -Organizacional
Lenguajes funcionales // Lenguajes formales	Funcional
Modelación de datos, incluyendo diagramas de entidad-relación, declaraciones de relaciones y de datos estructurados	Informacional
Modelación de objetos, incluyendo tipos de clases e instancias, jerarquía y herencia	Organizacional - Informacional
Modelación cuantitativa, incluyendo técnicas cuantitativas de la Investigación Operativa y la Dinámica de sistemas	Comportamiento
Redes de precedencia, incluyendo la modelación de la Dependencia del Actor	Comportamiento - Organizacional
Lenguajes de lógica temporal, CTL, TCTL	Funcional - Comportamiento

6. Conclusiones

En este artículo se ha descrito la modelización de las actividades, de los documentos de entrada y de salida, de las técnicas y los roles para el Proceso de Iniciación, Planificación y Estimación del MPSI. En este trabajo se ha utilizado una herramienta que permite considerar el aspecto dinámico temporal en el modelo propuesto. Mediante metaformalismos se analiza la evolución de una red considerando nociones temporales de presente y futuro, determinando cuáles son los posibles estados por los que debe atravesar la red, e identificar puntos en los que se pueda tomar decisiones para evitar que la red alcance algún estado crítico. Nótese que esta identificación de puntos de decisión considera otros factores tales como los problemas de dependencias entre los distintos subprocesos del proceso software, que exige centrarse en la modelación de la complejidad del flujo de comunicación e interacción para resolver estos problemas. En particular, en este trabajo, se establece claramente el orden de precedencia y simultaneidad de las actividades del subproceso considerado. Se logra precisar el orden y secuencialidad de las actividades involucradas en el proceso propuesto.

En cuanto a la aplicación de AToM^3, en particular el model checking, permite a partir de la simulación y análisis del comportamiento del subproceso de gestión modelizado, afirmar que las especificaciones efectuadas como resultado de los requerimientos del diseño son coherentes y consistentes, ya que el comprobador de modelos permite detectar los posibles fallos del modelo del sistema, efectuar las correcciones y garantizar la ausencia de errores.

Por último, se destaca la importancia de disponer de técnicas de modelización y de verificación formal integradas como herramientas fundamentales de apoyo a los métodos convencionales de análisis y diseño de sistemas, especialmente para aquellos sistemas cuyo fracaso significa grandes costos.

Referencias

[1] S. T. Acuña, A. de Antonio, X. Ferré, M. López y L. Maté, *The Software Process: Modelling, Evaluation and Improvement.* Handbook of Software Engineering and Knowledge Engineering. Vol. I (World Scientific, 2001) 193-237.

[2] S. T. Acuña, Capabilities-Oriented Integral Software Process Model. PhD Thesis, Universidad Politécnica de Madrid (2002).

[3] AToM^3 home page: http://moncs.cs.mcgill.ca/MSDL/research/projects/ATOM^3.html

[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] N. Bjorner, Z. Manna, H. Sipma y T. Uribe, "Deductive Verfication of Real Time System Using SteP". Technical Report STAN-CS-TR-98-1616, Computer Science Departament, Stanford University (Diciembre 1998) 40 pp.

[6] E. M. Clarke y J. M. Wing. "Formal Methods: State of the Art and Future Directions", ACM Computing Surveys, Vol 28, Nº4 (Dec. 1996), Carnegie Mellon University

[7] J. de Lara y H. Vangheluwe. "AToM^3: A Tool for Multi-Formalism Modelling and Meta-Modelling". In Proc. ETAPS/FASE'02. LNCS 2306, 174 - 188. Springer-Verlag. (2002)

[8] J. de Lara y H. Vangheluwe. "Computer Aided Multi-Paradigm Modelling to process Petri-Nets and Statecharts". ICGT'2002. LNCS 2505. 239-253. (2002)

[9] J. de Lara y G. Taentzer. "Automated Model Transformation and its Validation with AToM^3 and AGG". Diagrams'2004. Lecture Notes in Artificial Intelligence 2980. Springer. 82-198. (2004)

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

[11] E.A. Emerson y E.M. Clarke."Using branching-time temporal logic to synthesize synchronization skeletons". Science of Computer Programming. 2:241-266. (1982)

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

[13] E. Guerra y J. de Lara. "A Framework for the Verification of UML Models. Examples using Petri Nets". Proc. of Jornadas de Ingeniería del Software y Bases de Datos (2003)

[14] J. A. W. Kamp, On Tense Logic and the Theory of Order. PhD Thesis, University of California (1968).

[15] 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.

[16] 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.

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

[18] 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.

[19] Z. Manna y A. Pnuelli, *The Anchored Version of the Temporal Framework*. Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency (Springer Verlag, 1989) 201-284.

[20] Z. Manna y A. Pnuelli, "A hierarchy of temporal properties". *Proceedings of the 9th Annual ACM Symposium on* Principles *of Distributed Computing* (ACM Press, 1990) 377-408.

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

[22] S.-Y. Min y D.-H. 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.

[23] P. J. A. Morssink, Behaviour Modelling in Information Systems Design: Application of the PARADIGM Formalism. PhD Thesis, University of Leiden (1993).

[24] Unified Modeling Language (UML) 1.5 specification. (March 2003). Available at the OMG's home page: http://www.omg.org/UML