Modelos de Estructuras de Sistemas para Ambiente Integrado de Ingeniería Automática y Métricas Asociadas

Elizabeth Jiménez Rey[(1)] María Delia Grossi[(2)] Arturo Carlos Servetto[(3)]

Ramón García Martínez[(4)] Gregorio Perichinsky[(5)]

([(1)]ejimenez [(2)]mdgrossi [(3)]aserve [(4)]rgm [(5)]gperi)@mara.fi.uba.ar

Laboratorio de Bases de Datos y Sistemas Operativos

Departamento de Computación - Facultad de Ingeniería - Universidad de Buenos Aires

Paseo Colón Nº 850 - (1063) Buenos Aires - Argentina

Teléfono: (+54-11) 4343-0891 (int. 140/142)

Palabras Clave

Herramientas CASE, Escenarios de Sistemas, Modelación de Operaciones y Comportamiento de Sistemas, Máquinas de Estados Finitos, Eventos en un Sistema, Diseño de Sistemas Orientado a Eventos, Interfaces Orientadas a Objeto.

Resumen

El objetivo de este trabajo es especificar un modelo para representar la estructura dinámica de sistemas que, basado en el análisis por escenarios y casos de uso y en la clasificación de objetos de un sistema en objetos de aplicación y de interfaz, sintetiza la funcionalidad y el comportamiento de un sistema en una máquina de estados finitos por escenario. Además se proponen métricas para medir la complejidad de sistemas acordes al proceso de desarrollo.

El modelo resulta apto para la automatización del desarrollo de sistemas de oficina o de gestión tradicionales y reduce a un único diagrama por escenario toda la especificación del comportamiento y funcionalidad de un sistema, sintetizando los escenarios o casos de uso, la interacción entre objetos (diagramas de secuencia y de colaboración) y las secuencias de estado, propuestos por el UML (Unified Modeling Language) de Booch, Rumbaugh y Jacobson [Booch et al. 99].

Introducción

Los ingenieros de software aficionados siempre buscan algún método o herramienta sensacional cuya aplicación o uso asegure que el desarrollo de software sea trivial. Un rasgo distintivo de los profesionales de la ingeniería de software es que saben que no existe esa panacea. Los aficionados quieren seguir instrucciones en forma de receta; los profesionales saben que los enfoques directos del desarrollo suelen conducir a diseños fallidos, detrás de los cuales los desarrolladores pueden escudarse para evitar responsabilidades de decisiones iniciales erróneas. El ingeniero de software aficionado, o bien ignora la documentación por completo o bien sigue un proceso dirigido por la documentación, preocupándose más sobre el aspecto que presentan de cara al cliente esos productos de papel que sobre la sustancia que contienen. El profesional reconoce la importancia que tiene crear ciertos documentos, pero nunca lo hace a expensas de realizar sensibles innovaciones arquitectónicas [Booch 94].

La idea del proyecto que enmarca este trabajo es proponer un proceso de desarrollo estándar simple, que no imponga la realización de una multitud de diagramas que demoran el proceso y distraen recursos, y que requiera la máxima atención y especialidad en el análisis y el diseño, a partir de cuyos

diagramas se deriva directamente el código. Su propósito es identificar, diseñar, desarrollar e integrar los componentes de un ambiente integrado de desarrollo automático de sistemas a partir de especificaciones formales de alto nivel de abstracción. Se pretende lograr la generación de sistemas a partir de sólo dos modelos: el estático o de estructura de datos, basado en diagramas de clases, y el dinámico o funcional, basado en la especificación formal de operaciones en un álgebra relacional de objetos y en la teoría de autómatas finitos. El mantenimiento de los sistemas generados por la herramienta se efectuaría operando directamente sobre los modelos estático y dinámico, sin necesidad de recodificar ni de efectuar ingeniería inversa.

Un Modelo de Estructura Dinámica de Sistemas

Muy sintéticamente, se puede decir que una máquina de estados finitos es un modelo abstracto de máquina o sistema en función de sus *entradas y salidas* organizadas secuencialmente. Un sistema de información o parte de él se puede modelar entonces como un *circuito secuencial* cuyas entradas son algún tipo de información y sus salidas los resultados de procesar la información de entrada. La forma de procesar las entradas depende de las *operaciones* o procesos internos del sistema, que en el mismo modelo abstracto se pueden especificar declarativamente mediante algún mecanismo formal como el álgebra o los cálculos relacionales (aún en sistemas orientados a objetos, puesto que no es realmente significativo para la sintaxis de estos mecanismos que sean tuplas u objetos los elementos a procesar).

Las principales características de una máquina de estado son las siguientes [Bass et al 98], [Constantine, Lockwood 99], [Sommerville 96], [Grimaldi 89]:

1. La máquina sólo puede estar en un *número finito* de estados durante un período determinado. Tales estados se denominan *estados internos* “I” de la máquina y en todo momento la máquina conoce el estado interno en que se encuentra.
2. La máquina aceptará como entrada sólo un número finito de símbolos, que se conocen como *alfabeto de entrada* “E”.
3. Se determina una *salida* o un *estado siguiente* por cada combinación de entradas y estados de la máquina. El conjunto finito de todas las salidas posibles constituye el *alfabeto de salida* “S”.
4. Se supone que los procesos secuenciales de la máquina están *sincronizados* de alguna manera, y que la máquina opera en forma *determinista*, ya que la salida está determinada en todo momento por el total de entradas proporcionadas y por el estado inicial de la máquina.

Por tanto una *máquina de estados finitos* es un quinteto $M = < I, E, S, Fe, Fs >$ donde

I es el conjunto de estados internos de M; E el alfabeto de entrada; S el alfabeto de salida;

$Fe: I \times E \rightarrow I$ la *función del estado siguiente*, y $Fs: I \times E \rightarrow S$ la *función de salida*.

Así es que si la máquina se halla en el estado interno i en un momento t y se introduce e en ese momento, entonces la salida en t será Fs(i, e) a la cual le sigue una transición de la máquina de manera que en el momento t+1 estará en el estado Fe(i, e).

Para aplicar este modelo de representación de comportamiento a un sistema informático, se suelen considerar los *eventos* que pueden afectar al sistema o acaecer en él como desencadenantes de transiciones de estados [Martin 94]. Pero las definiciones de estados, de las funciones o procedimientos que hacen que el sistema pase de uno a otro, y de los mismos eventos desencadenantes son subjetivas del diseñador, y así un mismo sistema puede descomponerse en más o menos estados según el criterio que se emplee.

Dada la profusión de herramientas y bibliotecas de clases para el desarrollo de interfaces en el marco de la Orientación a Objetos, y el empleo de éstas en la Programación Orientada a Eventos, surge la necesidad de establecer técnicas de especificación que combinen estos principios para formalizar y documentar el desarrollo de sistemas. Para ello se puede considerar una adaptación de las máquinas de estados finitos a las particularidades de los sistemas informáticos orientados a eventos.

Como la programación orientada a eventos se sustenta en la orientación a objetos con la técnica de manejo de interfaces MVC (Model-View-Controller), una forma de aplicar como mecanismo de modelación conceptual de comportamiento a las máquinas de estados finitos sería asociar directamente cada interfaz distinta de un sistema con un estado del mismo. Esto tiene la virtud de estandarizar el criterio de descomposición de los sistemas en funciones o procedimientos, ya que optando por este modelo la única forma posible es considerar módulos de software (métodos) asociados a cada objeto de interfaz (view o adaptor) y que operen con los objetos de aplicación que le correspondan (model). De la detección de los eventos y la invocación de los módulos o métodos se encargan los objetos controladores (controllers), a cargo del lenguaje de programación.

La diferencia fundamental de las máquinas de estados finitos adaptadas para la especificación de comportamiento de sistemas respecto de sus parientes matemáticas, radicaría en la concepción y especificación de las entradas y salidas. En las máquinas formales, todo símbolo del alfabeto de entrada debe ser aceptado por todo estado para las funciones de estado siguiente y salida, aún cuando el estado siguiente sea el mismo de partida o la salida sea nula.

En las máquinas de sistemas informáticos, los alfabetos de entrada pueden ser considerablemente extensos y complejos y para cada estado sólo pueden ser pertinentes un número muy pequeño de elementos de ese conjunto en comparación con su cardinalidad; para el resto de los elementos el estado siguiente sería el mismo y la salida nula. Con las salidas sucede algo similar en cuanto a la complejidad y a la dependencia del estado. En realidad, en un sistema informático las entradas y salidas se asocian a los módulos o métodos componentes, que en este modelo corresponden a razón de uno por cada transición de estado determinada por Fe-Fs.

Por tanto, para cada estado sólo se consideran las transiciones asociadas a las entradas significativas, y las entradas y salidas asociadas a cada transición se pueden sintetizar en la especificación del proceso o función inherente. Para esto, la forma de representación óptima resulta ser la gráfica, en la que se denotan estados o interfaces por medio de círculos y transiciones por medio de arcos dirigidos. A cada arco se puede asociar un nombre descriptivo del proceso o función responsable de la transición, que conlleva implícitas una entrada y una salida.

Como los objetos de interfaz en un sistema orientado a eventos se pueden clasificar esencialmente en menús, formularios editables e informes o reportes wysiwyg (what you see is what you get: lo que ve es lo que obtiene) no editables, y a los formularios en simples (de instancia), continuos y tabulares (ambos de clase o conjunto de instancias), las transiciones entre estos tipos de objetos difieren considerablemente en naturaleza.

Una transición hacia una interfaz tipo menú estará siempre representada por un simple método de control, cuya entrada corresponderá a la pulsación de una tecla o a la selección de una opción y cuya salida será el mismo objeto de interfaz. En cambio una transición hacia una interfaz de alguno de los otros tipos o que los comprenda, requerirá un proceso más elaborado, sobre objetos de aplicación. Es así que la primer categoría de transiciones no requiere ninguna especificación explícita, basta con denotarlas tan sólo con un arco; mientras que la otra categoría sí requiere de un proceso explícito, que deberá designarse con un nombre o número que puede ser relativo al estado. Luego mediante convención ad-hoc, se pueden especificar formalmente dichos procesos consignando entradas y salidas en forma explícita.

De la manera anteriormente detallada, los diagramas de representación de comportamiento resultan de fácil comprensión y no se saturan de especificaciones que pueden distraer o escapar del ancho de banda de la mente humana. La especificación de los procesos o funciones asociados a cada transición se realiza, por ejemplo, mediante el álgebra relacional en un documento a parte o mediante controles sobre cada arco, si se emplea la técnica descrita en alguna herramienta de software.

El que sigue es un ejemplo simple de aplicación de la técnica propuesta, sobre un sistema para un servicio de consultorios externos de un hospital. Primero se especifica la estructura estática de la información mediante un diagrama de clases, y luego el diagrama de estructura dinámica en el método que se propone.

Diagrama de Estructura Estática

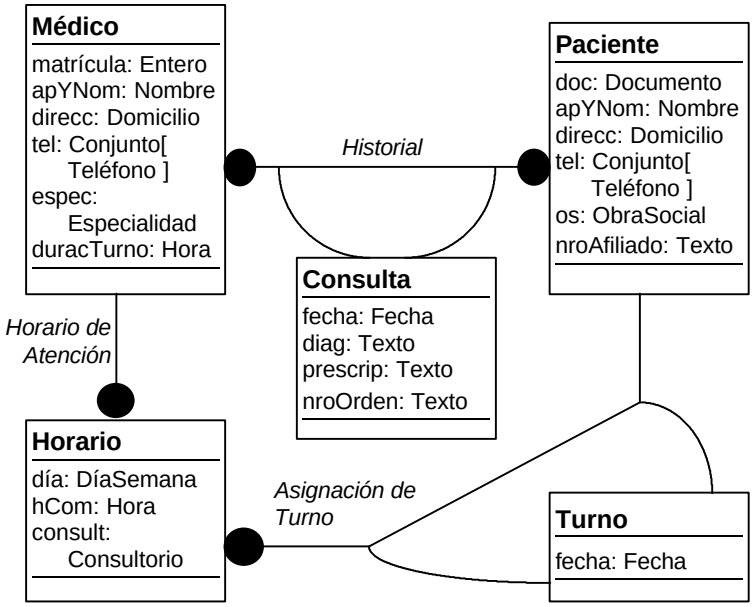
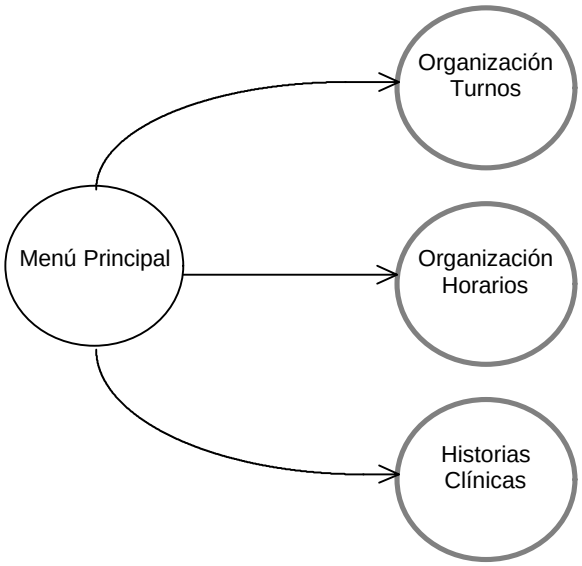


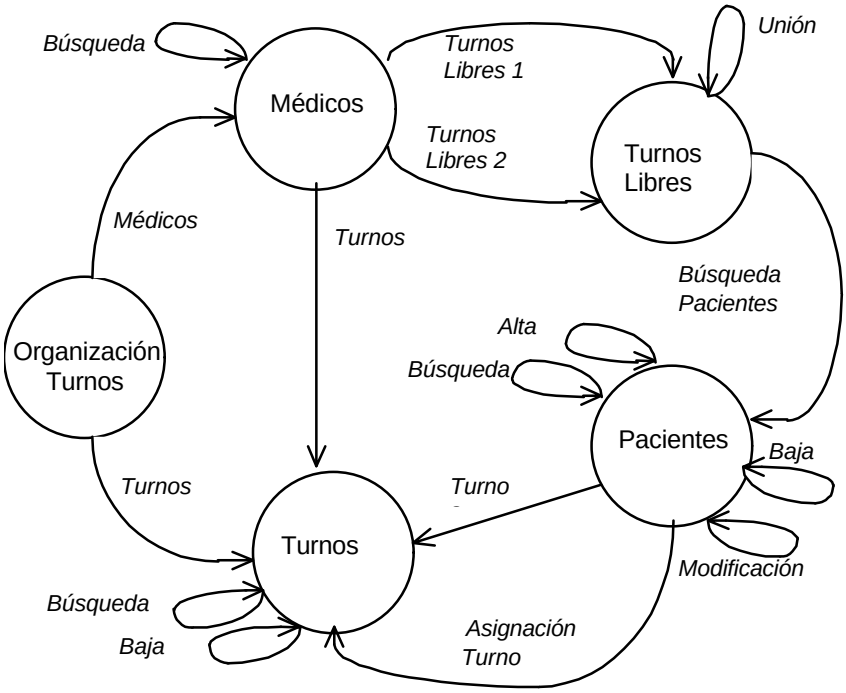
Diagrama de Estructura Dinámica



Este primer diagrama indica que el sistema consta de un menú principal que lo divide en tres escenarios principales: el de Organización de Turnos, el de Organización de Horarios y el de Historias Clínicas. La circunstancia de que un estado es un escenario está denotada por el grosor de la circunferencia, lo cual significa que ese estado es una máquina de estados finitos que se especifica en otro diagrama subyacente. A los escenarios o casos de uso se les puede asociar grupos de usuarios con privilegios de acceso.

Como el estado inicial del diagrama anterior es un menú que conduce a submenús, no se especifican procesos o funciones para las transiciones correspondientes. Se asume la reversibilidad de toda transición por defecto (no se denota), por ejemplo mediante la tecla [Escape].

A continuación se incluye el diagrama de uno de los escenarios principales, a los efectos de redondear el ejemplo y visualizar transiciones hacia objetos de interfaz de tipo formulario, que requieren la designación y especificación formal posterior de los procesos correspondientes.



El enfoque de diseño de este escenario establece dos opciones principales para la organización de turnos, a cargo de uno o más actores que realizan funciones de secretaría del Servicio de Consultorios: convienen turnos con pacientes que pueden llamar telefónicamente o los piden en forma personal, y hacen pasar a los pacientes de cada médico según el turno que les toque.

La primera opción, para gestionar turnos, que se representa a partir del arco superior desde la primera interfaz del escenario, exige determinar en primer lugar para qué médico será la consulta a establecer (o de qué médico el turno a cancelar). Para esto desde el menú principal de Organización de Turnos se pasa a una interfaz tipo formulario (la clase de formulario a emplear puede decidirse en un refinamiento posterior) que en principio puede tener asociados a todos los médicos que actualmente atienden en el Servicio o a los de la especialidad que se determine con un parámetro de entrada o directamente al que se busque por apellido.

Una vez en el formulario de Médicos se puede seleccionar definitivamente al profesional para quien se concertará la consulta y pasar luego a Turnos Libres para ese médico, o, si el objetivo es cancelar un turno ya establecido, pasar a Turnos (de ese médico) para dar de baja al que corresponda. No se incluyen las típicas operaciones ABM para Médicos, dado que se consideran reservadas al escenario

de Organización de Horarios, en el que otros actores se encargan además de mantener la grilla de horarios de atención de los profesionales y asignarles consultorios.

Y así se puede seguir interpretando todo el diagrama, en *forma navegacional*. Las operaciones declaradas en cada arco de transición entre interfaces como se dijo antes se pueden especificar formalmente en otro documento ad-hoc o, si el método se implementa en una herramienta CASE, seleccionando un arco con el ratón se puede ingresar a un texto asociado a él en donde formalizar la operación mediante el álgebra relacional o ya directamente en un lenguaje estructurado de consultas (SQL), adaptados al modelo de datos que se emplee.

A propósito de lo anterior, para completar la propuesta con un formalismo de especificación de operaciones, a continuación se sugiere una adaptación del álgebra relacional aplicada al ejemplo en desarrollo.

Transiciones desde Organización Turnos (menú)

Médicos

$\sigma_{@espec \wedge @apYNom}$ (Médico)

donde "@" en la condición de selección "@espec ∧ @apYNom" significa que se debe ingresar por teclado (o lista de selección) una especialidad y un apellido (o parte) como patrones de matching para los respectivos atributos. Si no se desea filtrar ningún médico (ya esto es materia de implementación), para ambos atributos se podría ingresar el comodín " * " (significa ∀); si se desean todos los médicos de una especialidad se puede ingresar por ejemplo el par de valores "Traumatólogo" y " * "; si se desea una búsqueda aproximada por apellido, " * " para la especialidad y por ejemplo "H*" para médicos con apellidos que comienzan con H; etc.

Turnos

$\Pi_{\text{Paciente.apYNom, Médico.apYNom, Horario, fecha}}$ ($\sigma_{@\text{Médico.apYNom} \wedge @\text{Paciente.apYNom} \wedge @\text{fecha} \wedge @\text{hCom}}$ (Médico $\times_{\text{Horario de Atención}}$ Horario $\times_{\text{Asignación de Turno}}$ Paciente))

donde el símbolo × representa a los objetos vinculados por las asociaciones establecidas como subíndice (la clase asociación Turno está implícitamente comprendida).

Transiciones desde Médicos (formulario con esquema de clase Médico)

Búsqueda

Igual que *Médicos* desde Organización Turnos

Turnos Libres 1

Esta operación requiere la creación de instancias de una clase temporal TL (turnos libres) a los efectos de establecer qué turnos no están ocupados en una fecha o período determinado (todos los turnos posibles en el período menos todos los dados):

TL := $\Pi_{\text{Médico.apYNom, Horario, fecha}}$ (($\sigma_{\&\text{Médico}}$ (Médico) $\times_{\text{Horario de Atención}}$ Horario $\times_{\text{día=díaSemana(fecha)}}$ { (F1 := @Turno.fecha) hasta (F2 := @Turno.fecha) }) − ($\sigma_{\&\text{Médico}}$ (Médico) $\times_{\text{Horario de Atención}}$ Horario $\times_{\text{Asignación de Turno}}$ Turno))

donde "&Médico" significa que sólo se seleccionarán objetos vinculados a la instancia de Médico activa en el estado o interfaz de partida, y dentro de las llaves de conjunto se especifica un intervalo de fechas como instancias de una clase anónima temporal creada ad-hoc (la alusión a la clase Turno para denotar la lectura del valor sirve al

sólo efecto de determinar el tipo del valor); y si se quisieran los turnos libres de un sólo día bastaría con especificar un intervalo con la misma fecha como inicio y fin.

Turnos Libres 2

Esta operación sería similar a *Turnos Libres 1* con la sola diferencia de que en vez de procesarse los turnos libres del médico seleccionado en la interfaz Médicos (operación a partir de una instancia), se procesan los turnos de todos los médicos de la interfaz (operación a partir de un parte de una extensión de clase): se reemplaza "$\sigma_{\&Médico}$ (Médico)" por "Médico" en sus dos ocurrencias.

Turnos

$\Pi_{Paciente.apYNom,\ Médico.apYNom,\ Horario,\ fecha}$ ($\sigma_{\&Médico \wedge @Paciente.apYNom \wedge @fecha \wedge @hCom}$ (Médico $\times_{Horario\ de\ Atención}$ Horario $\times_{Asignación\ de\ Turno}$ Paciente))

Transiciones desde Turnos Libres (formulario con esquema Médico.apYNom, Horario, fecha)

Unión

TL := TL $\cup$ $\Pi_{Médico.apYNom,\ Horario,\ fecha}$ (($\sigma_{\&Médico}$ (Médico) $\times_{Horario\ de\ Atención}$ Horario $\times_{día=díaSemana(fecha)}$ { (F1 := @Turno.fecha) hasta (F2 := @Turno.fecha) }) − ($\sigma_{\&Médico}$ (Médico) $\times_{Horario\ de\ Atención}$ Horario $\times_{Asignación\ de\ Turno}$ Turno))

donde se supone que la o las fechas que se ingresen al efectuar esta operación serán posteriores a las ya establecidas en TL.

Búsqueda Pacientes

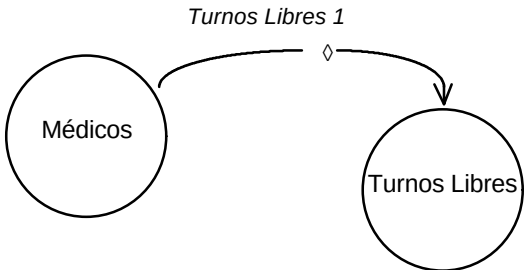
$\sigma_{@nroAfiliado \wedge @apYNom}$ (Paciente)

Y así se sigue con todas las interfaces, siempre teniendo en consideración que todos los objetos seleccionados en las interfaces previas en la navegación, sean permanentes o temporales, se encuentran disponibles denotándolos con el operador "&". Si se prefija a una clase con este operador se estará aludiendo al IDO del objeto seleccionado de esa clase, y si se prefija un atributo, se estará aludiendo al valor del atributo. De esta propiedad de *memoria de estados pasados* deberá hacerse uso cuando se formalice la asignación de turno al paciente seleccionado en Pacientes, puesto que habrá que incorporar a la clase Turno la fecha de la instancia de TL seleccionada en Turnos Libres, vinculada a la instancia de Paciente y de Horario respectivas.

En los sistemas orientados a eventos es común establecer condiciones de control para que puedan producirse algunos eventos. Si bien en el ejemplo desarrollado no apareció ninguna, se pueden considerar para varias transiciones, por caso para todas las posibles desde Médicos: para que se produzca cualquiera de las transiciones posibles es menester que haya algún médico seleccionado en Médicos. La condición de control para cada transición, entonces, podría especificarse asociada a la formalización de las operaciones respectivas, como una precondición con la fórmula

&Médico $\neq$ Nulo

Y en cuanto a la denotación de las condiciones de control en el diagrama de comportamiento, si se juzgara pertinente que quedaran indicadas allí, se pueden distinguir los arcos de transición que tengan asociada una con un símbolo "◊" en su trayectoria, por ejemplo:



Como se puede ver de lo expuesto, el modelo resulta apto para la automatización del desarrollo de sistemas de oficina o de gestión tradicionales y reduce a un único diagrama por escenario toda la especificación del comportamiento y funcionalidad de un sistema, sintetizando los escenarios o casos de uso, la interacción entre objetos (diagramas de secuencia y de colaboración) y las secuencias de estado, propuestos por el UML (Unified Modeling Language) de Booch, Rumbaugh y Jacobson. También resulta apto para su empleo en ambientes CASE de desarrollo integral de aplicaciones (con generación automática de código) y posibilitaría cambiar el comportamiento o adaptar un sistema cambiando las especificaciones en los modelos conceptuales [Servetto 97][Servetto 98].

Métricas

Métrica de la Estructura Estática

Como el modelo estático es conceptual, es decir, de alto nivel de abstracción, y los mecanismos de abstracción en la modelación de datos son la clasificación, la agregación y la generalización [Batini et al. 91], el uso de cada uno de estos mecanismos debe considerarse en la medición de la complejidad de un diagrama en cuanto a la estructura de sus datos [Fenton, Pfleeger 97][Pfleeger 98].

La **abstracción de clasificación** se usa para definir un concepto como una *clase* de objetos de la realidad que se caractericen por propiedades comunes. Una **abstracción de agregación** define una nueva clase a partir de un conjunto de otras clases o tipos que representan sus partes componentes. Una **abstracción de generalización** define una relación de subconjunto entre los objetos o entidades de dos o más clases.

Por lo tanto, los parámetros que se pueden considerar para una medición de complejidad serían la cantidad de clases C (clasificación), la cantidad total de atributos A (agregación) y la parcial de atributos complejos CA (clasificación y agregación), la cantidad total de relaciones o asociaciones entre clases R (agregación) y la parcial de asociaciones como clases CR (clasificación y agregación) y la cantidad de clases que son subconjuntos de generalización G.

La complejidad estática o de datos del sistema puede medirse como

$$C + C * VA + C * VR + C * VG = C * (1 + VA + VR + VG)$$

donde VA, VR y VG son factores que acrecientan la complejidad en función de los atributos, relaciones y generalizaciones, y se calculan

$$VA = C / A * (1 + CA / A) \qquad VR = R / C * (1 + CR / R) \qquad VG = G / C$$

En el caso de estudio anterior, la medida de complejidad del modelo estático del sistema se evalúa con C=5, A=20, CA=10 (definidos con tipos especiales), R=3, CR=2 y G=0. Por tanto VA=5/20*(1+10/20)=0.375, VR=3/5*(1+2/3)=1 y VG=0/5=0. Y la complejidad estática o de datos resulta 5 * (1 + 0.375 + 1 + 0)= 11.875. Desde luego estos valores deben interpretarse en el marco

de una escala predeterminada que de sentido a la medición, y cuyos extremos deben ser cuidadosamente determinados a partir de casos concretos.

Métrica de la Estructura Dinámica

Como del modelo de estructura dinámica surge la complejidad de un sistema tanto en su faz de comportamiento, en términos de sus interfaces, como en su faz funcional, en términos de sus operaciones, cabe considerar estos dos aspectos separadamente.

La complejidad de un sistema respecto del diseño de interfaces puede mensurarse considerando la cantidad total de interfaces I, más la cantidad de funciones asociadas F, de la siguiente manera

$$I * (1 + I / F)$$

donde el segundo factor acrecienta la complejidad según la cantidad de funciones u operaciones. Por caso, la complejidad de diseño del subsistema de Organización de Turnos del caso de estudio sería entonces 5*(1+5/16)= 6.5625.

La complejidad funcional de un sistema en este ambiente de desarrollo se puede medir ponderando cada función en términos de las operaciones relacionales de su especificación formal, y considerando como factor de acrecentamiento de complejidad el nivel de composición de cada operación. Aunque en el caso de estudio cada función se especifica con una única operación relacional compuesta, para los casos de transacciones complejas se considera una extensión del álgebra relacional de objetos para hacerla computacionalmente completa (se agregan estructuras de control algorítmico, navegación de tablas y vistas –cursores– y uso de variables), en los que la especificación se transforma en un guión o secuencia de operaciones.

La complejidad total de un sistema podría entonces calcularse como una combinación lineal de los tres valores especificados. El valor resultante, enmarcado en una escala comparativa adecuada, puede servir como medida del esfuerzo que insumió su desarrollo considerados todos los aspectos del proceso, y servir como base para la valoración del producto.

Conclusiones

La generación automática de sistemas a partir del diseño gráfico de estructuras estáticas en un diagrama, y de estructuras dinámicas en otro, apoyadas por la especificación formal de operaciones en un lenguaje estándar, representa una simplificación del proceso de desarrollo de software que además de ahorrar tiempo y recursos para programación y pruebas implica la autodocumentación de los productos.

El mantenimiento de los productos de software también se automatiza por medio de la herramienta mediante correcciones en especificaciones formales de alto nivel de abstracción, eliminando la necesidad de la ingeniería reversa para el mantenimiento de la documentación.

El hecho de que la herramienta se base en especificaciones formales absolutamente independientes de modelos y lenguajes específicos de implementación permitiría a cualquier profesional en informática operarla casi sin entrenamiento. Esto hace que la herramienta sea de fácil adopción por instituciones de desarrollo de software, ya que esto no implica costos adicionales y la inversión inicial permitiría retornos inmediatos.

Asimismo, la manipulación de conceptos abstractos representados gráficamente acerca a los usuarios finales al proceso de desarrollo, permitiéndoles participar, seguir y controlar la evolución de los sistemas, pudiendo formalizarse procedimientos de acreditación o certificación de avances que despejen dudas e incertidumbres y fortalezcan la confianza de los usuarios respecto a la observación de requerimientos y la calidad de los productos.

Las métricas de complejidad que se incluyen en la herramienta facilitan también la determinación del valor de los sistemas producidos con ella, con criterio documentado y uniforme.

Trabajos Futuros

Trabajos posteriores a esta propuesta preliminar incluirán la contrastación de la técnica desarrollada contra las técnicas actualmente en uso y la especificación de una metodología integradora y abarcativa del proceso de desarrollo de sistemas, sustentable por un ambiente de desarrollo automático, que incorpore el método definido.

Así también, la modelación de la generación automática de código a partir de modelos gráficos integrados, de manera que los cambios al modelo de comportamiento propuesto y/o al modelo de estructura de un sistema, operen automáticamente sobre el mismo sin necesidad de retraducir código.

También se proyecta formalizar métricas para el control de calidad, que permitan probar automáticamente los diagramas conceptuales de estructura y comportamiento de los sistemas, para obtener indicaciones respecto a distintas propiedades de los productos de software y eventualmente efectuar optimizaciones en consecuencia.

Referencias

[Bass et al 98] Software Architecture in Practice. Bass,Clements and Kazman, Addison-Wesley, 1998.

[Batini et al. 91] Diseño Conceptual de Bases de Datos: un enfoque de entidades e interrelaciones. Batini, Navathe, Ceri. Addison Wesley 1991.

[Booch 94] Object-Oriented Analysis and Design with Aplications. Second Edition. Booch, G. Addison-Wesley, 1994.

[Booch et al. 99] The Unified Modeling Language - User Guide. Booch, Grady; Rumbaugh, James; Jacobson, Ivar. Addison-Wesley, 1999.

[Constantine, Lockwood 99] Software for use A Practical Guide to Models and Methods of Usage Centred Design. Constantine, L and Lockwood, L, Addison-Wesley, 1999.

[Fenton, Pfleeger 97] Software Metrics. Fenton, N and Pfleeger, S, PWS Publishing Company 1997.

[Grimaldi 89] Matemáticas discreta y combinatoria. Introducción y aplicaciones. Grimaldi, Ralph P. Addison-Wesley Iberoamericana, 1989.

[Martin 94] Análisis y Diseño Orientado a Objetos. James Martin y James J. Odell; Prentice Hall Hispanoamericana, 1994.

[Pfleeger 98] Software Engineering Theory and Practice. Pfleeger, S, Prentice Hall 1998.

[Servetto 97] Herramienta CASE Orientada A Objetos Para Automatización De Oficinas. Servetto, Arturo C. CACIC 97 (III Congreso Argentino de Ciencias de la Computación), La Plata 29-9-97 al 4-10-97, Pág. 71 Abstracts.

[Servetto 98] Una Herramienta ICASE para el Desarrollo y Mantenimiento de Software Orientado a Objetos. Servetto, Arturo C. IV Congreso Internacional de Ingeniería Informática, FIUBA, 16 y 17 de abril de 1998, pp 311 a 325.

[Sommerville 96] Software Engineering. Sommerville, I, Addison Wesley, 1996.