Análisis de secuencias discretas para la detección de Patrones de Diseño de Software

Juan Francisco Silva Logroño, Luis Berdún, Marcelo Armentano[1], Analia Amandi[1]

Instituto de Investigación ISISTAN, Fac. de Cs. Exactas, UNCPBA
Campus Universitario, Paraje Arroyo Seco, Tandil, 7000, Argentina
[1] también CONICET, Consejo Nacional de Investigaciones Científicas y Técnicas, Argentina
{juanf.silval}@gmail.com; {lberdun, marmenta, amandi}@exa.unicen.edu.ar

Abstract. Los patrones de diseño de software son una herramienta que permite re utilizar las experiencias previas en el diseño actual. Los diseñadores inexpertos en dicha área se ven obligados a leer catálogos de patrones a fin de nutrirse de este conocimiento, perdiendo el aprendizaje que da la práctica. Partiendo de este problema, en el presente artículo se propone la utilización de Modelos de Markov de Orden Variable para brindar, a partir de un conjunto acotado de actividades desarrolladas por un diseñador y una base de conocimiento preestablecida (plan corpus), toda aquella información necesaria para que un agente de Interfaz pueda aconsejar a un usuario durante el proceso de diseño de software.

Keywords: Agentes de Interfaz, Modelos de Markov, Patrones de Diseño.

1 Introducción

Los Agentes de Interfaz [1] surgen con la finalidad de proveer a los usuarios asistencia proactiva y reactiva de manera personalizada en el uso de una aplicación de software. Para proveer asistencia, los agentes de interfaz se basan tanto en los intereses, preferencias, prioridades y necesidades demostradas por el usuario a lo largo de su interacción con el agente. Sin embargo, la tarea de un agente de interfaz no reside únicamente en aprender las preferencias y hábitos demostrados por el usuario durante el uso de la aplicación, sino que debe considerar cual es el objetivo del usuario previo a comenzar a interactuar con este.

Para la integración del servicio de recomendación, se deben tener en cuenta dos factores primordiales: por un lado el foco de atención del usuario con respecto a la actividad que está realizando, y por el otro la incertidumbre ante el objetivo a cumplir. Con estas premisas es que surge la necesidad de crear un agente capaz de detectar de manera temprana el objetivo del usuario, de manera tal que la asistencia proporcionada por el agente se adecue al contexto de la tarea que está realizando el usuario.

A lo largo de los años los agentes de interfaz han sido utilizados en numerosos tipos de aplicaciones, desde editores de texto hasta servicios de mensajería instantánea. Sin embargo, en muchas ocasiones la inclusión de agentes de interfaz dentro de aplicaciones convencionales ha generado el rechazo de los usuarios. Esto último debido a las interrupciones innecesarias con observaciones o sugerencias incorrectas, debido a una mala detección del objetivo del usuario; un ejemplo claro de esto es el agente de asistencia del Microsoft Word 97 [2]. En relación a lo anterior y debido a la dificultad de predecir el objetivo del usuario durante el diseño de un sistema de software, es que la utilización de

agentes de interfaz dentro de herramientas CASE[1] ha sido un campo poco explorado. En varias de las aplicaciones que buscan este cometido [3], los medios utilizados por el agente para aconsejar al usuario exigen de un tiempo computacional del que generalmente no se dispone en una aplicación de estas características.

La finalidad de este trabajo es analizar las secuencias de acciones realizadas por un usuario al momento de utilizar una herramienta CASE, con el fin de lograr una detección temprana del patrón de diseño que se está intentando modelar. Para ello se propone la utilización de Modelos de Markov de Orden Variable, tanto para disminuir los tiempos computacionales necesarios para identificar la meta buscada por el usuario como para formular un conjunto de posibles acciones a seguir. Dichos modelos permitirán al agente de interfaz generar recomendaciones en base a cada una de las acciones realizadas por el usuario.

El trabajo se organiza en 7 secciones. En la Sección 2 se realiza una breve reseña sobre los Patrones de Diseño que se utilizan con mayor frecuencia en la actualidad. En la Sección 3 se presenta una introducción a los Modelos de Markov de Orden Variable y su aprendizaje, y en la Sección 4 se describe su utilización para la detección de patrones. Luego, en la Sección 5 se instancia a los Modelos de Markov para su utilización con los Patrones de Diseño anteriormente nombrados, desarrollando uno a uno los aspectos que deben tenerse en cuenta al momento de combinarlos. En la Sección 6 se presentan los resultados experimentales obtenidos. En la Sección 7 se analizan trabajos relacionados al propuesto y, para finalizar, en la Sección 8 se la presentan las conclusiones del trabajo junto a algunas observaciones de posibles trabajos futuros.

2 Patrones de diseño de software

Un patrón de diseño describe una estructura recurrente de componentes que se comunican para resolver un problema general de diseño en un contexto particular. Nomina, abstrae e identifica los aspectos claves de una estructura de diseño común, lo que los hace útiles para crear un diseño orientado a objetos reutilizable. Identifica las clases e instancias participantes, sus roles y colaboraciones y la distribución de responsabilidades [4].

Tabla 1. – Patrones de diseño divididos en sus 3 categorías.

Categoría	Creacionales	Estructurales	De Comportamiento
Patrones	Abstract Factory, Builder, Factory Method, Prototype, Singleton	Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy	Chain of Responsability, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor

Los patrones de diseño surgen con el fin de abstraer soluciones a problemas comunes y poder repetirlas ante un problema similar las veces que sean necesarias. Con el fin de popularizar estas técnicas y de formar un lenguaje estandarizado entre los miembros del área, Eric Gamma junto a un grupo de diseñadores propuso en 1995 el primer catálogo de patrones de diseño. En este se presentaban 23 patrones de diseño divididos en tres categorías generales (Tabla 1): Creacionales, Estructurales y de Comportamiento. Cada una

[1] Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador.

de estas ofrecía soluciones generales a los posibles problemas que podían presentarse durante la tarea de diseñado, debiendo cada interesado en el área instanciarlas para su problema particular y utilizarlas de la manera más propicia.

3 Modelos de Markov de Orden Variable

Los modelos de Markov, son una técnica utilizada para modelar secuencias de acciones observadas a lo largo del tiempo. Básicamente, una cadena de Markov es un proceso estocástico que cumple con la propiedad de Markov. Ésta estipula que dado el estado actual, los estados futuros son independientes de los estados pasados. En otras palabras, la descripción del estado actual posee toda aquella información que puede llegar a influenciar en la futura evolución del proceso. En cada paso, el sistema tendrá dos opciones: mantenerse en el mismo estado o avanzar al próximo, de acuerdo a una distribución de probabilidad contenida en el estado actual. Los cambios de estado son llamados transiciones, y la probabilidad asociada a cada uno de los posibles cambios de estado, probabilidad de transición.

Las cadenas de Markov de orden fijo son una extensión natural en la que el próximo estado depende de un número fijo de estados anteriores. Pese a que esta extensión resulta beneficiosa en muchos dominios, presenta varios inconvenientes. Primero, solo los modelos que poseen un orden muy pequeño son aplicables en la práctica debido al crecimiento exponencial del número de estados para los modelos de alto orden. Segundo, en el caso de las secuencias de acciones llevadas a cabo por un usuario para alcanzar una meta, la probabilidad de la próxima actividad a realizar, no siempre está determinada por el mismo número fijo de acciones previas. Por lo general existe una cantidad variable de pasos que determina cuál será la próxima acción a realizar por el usuario.

Los modelos de Markov de Orden Variable (VOM por sus siglas en ingles – Variable Order Markov), surgen como una solución al crecimiento exponencial de los estados causado por el incremento del orden en los modelos. En estos modelos el número de variables aleatorias que condicionan la distribución de probabilidades está ligado al contexto que se está observando. Estos modelos plantean que en una situación real existe cierta relación entre los estados, representada por el contexto, donde algunos de los estados pasados no condicionaran a los estados futuros, obteniendo así una reducción en el numero de parámetros que conformaran al modelo [5].

Los algoritmos encargados de detectar a los modelos VOM sobre un alfabeto $\sum$, buscan crear un autómata finito probabilístico (PFA) que pueda modelar esta compleja secuencia de datos. A comparación de los modelos de Markov de orden m, que buscan estimar distribuciones condicionales de la forma $Pr(\sigma|s)$, con $s \in \sum^n$ y $\sigma \in \sum$, los Algoritmos de VOM aprenden dicha distribución condicional, donde la longitud del contexto |s| varía en función de la información estadística disponible de los datos de entrenamiento proporcionados. De esta manera, los modelos VOM proveen los mecanismos necesarios para detectar dependencias Markovianas, independientemente de cuál sea la longitud de la información observada.

Ron et al. planteo un algoritmo para el aprendizaje de modelos VOM desde una fuente de información [6] y Armentano [7] extendió dicho algoritmo para que trabajase de manera incremental, actualizando los modelos a la vez que se iba incorporando información. Este modelo es descripto utilizando una extensión del PFA, al cual decidió llamarse Autómata Probabilístico de Sufijos (PSA) y que será modelado en base a un Árbol Probabilístico de

Sufijos (PST). Este último modela el conjunto mínimo de sub secuencias (de longitud variable) necesarias para modelar a la fuente de información estadística.

A continuación se puede observar cómo se calcula el valor de probabilidad asociado a cada una de las transiciones. Sea $e^1, e^2, \dots, e^m$ un conjunto de ejemplos de entrenamiento de tamaño *m* y cuyos símbolos pertenecientes al alfabeto $\sum$. La longitud de cada uno de estos ejemplos está dada por l_i , es decir $e^i = e_1^i, e_1^i, \dots, e_1^i \forall e_j^i$. La probabilidad empírica de una secuencia *s* de longitud l es calculada en base a la *Ecuación 1*.

$$\tilde{P}_{(s)} = \frac{\sum_{i,j} \chi_s^{i,j}}{\sum_{i\ s.t\ l_i \geq |s|} (l_i - (|s|-1))} \quad \textbf{(1)}$$

$$donde\ \chi_s^{i,j} = \begin{cases} 1\ si\ s_1, s_2, \dots, s_i = e_j^i, e_{j+1}^i, \dots, e_{j+(l_i-1)}^i \\ 0\ en\ otro\ caso \end{cases}$$

El numerador representa el número de veces que la secuencia *s* fue observada en cada ejemplo de entrenamiento, mientras que el denominador representa el máximo número de ocurrencias posibles de cualquier secuencia de igual longitud.

La probabilidad condicional empírica de observar un símbolo σ a continuación de una secuencia *s* es calculada en la *Ecuación 2*.

$$\tilde{P}(\sigma|s) = \frac{\tilde{P}(s.\sigma)}{\tilde{P}(s)} \quad \textbf{(2)}$$

El Algoritmo de entrenamiento se encarga de construir un árbol probabilístico de sufijos donde cada nodo estará etiquetado con una cadena de caracteres de longitud inferior a un valor máximo *L*. De tal manera se mantendrá registro del número de oportunidades en que cada símbolo σ fue observado después de cada contexto *s*. Luego son calculados los valores de probabilidad de cada una de las transiciones a partir de la *Ecuación 2*. Posteriormente se realiza un proceso de poda el cual elimina todos aquellos nodos cuyo valor de predicción sea similar a otros que modelen contextos más acotados. También elimina los nodos que correspondan a sub secuencias raramente observadas y que predigan acciones con un valor de probabilidad inferior a un parámetro dado (γ_{min} , la probabilidad mínima que puede ser asignada en un PST a las acciones detectadas en un contexto dado). Todos estos esquemas de poda son controlados por un conjunto de parámetros del algoritmo de aprendizaje. Para finalizar se aplica una técnica de suavizado (*smoothing*), la cual busca contemplar todas las acciones que pueden no haberse observado en los contextos planteados, pero que pueden llegar a ocurrir cuando se utiliza al PST como medio de predicción. El algoritmo recolecta una masa de probabilidad equivalente a $|\sum|_{\gamma_{min}}$ y la distribuye entre todas las tareas del contexto.

Luego, se obtiene una estructura del tipo PSA equivalente al PST creado, que será capaz de asignar valores de probabilidad a una secuencia de acciones observadas en tiempo lineal a la longitud de las mismas. Cada uno de estos PSA modela la secuencia de acciones necesarias para alcanzar el diseño de cada uno de los patrones del catálogo de [4].

4 Detección de patrones utilizando Modelos de Markov de Orden Variable

Una vez que el agente posee un conjunto de modelos creados en base a las posibles acciones que debe realizar un usuario para modelar un patrón de diseño en una herramienta

CASE, el mismo será capaz de detectar el patrón que está intentando modelar el usuario mientras interactúa con la herramienta.

Para realizar la detección de los patrones de diseño, el agente posee un PSA por cada uno de los posibles patrones. De esta manera, el agente será capaz de realizar el seguimiento de múltiples metas en paralelo, sin dejar de lado a ninguna de ellas, pudiendo ser unas más probables que otras tras una acción, pero no por ello menos probable en próximas instancias (con el pasar de las acciones). Siendo así, el proceso de detección de patrones de diseño consistirá en clasificar la acción observada en uno de los posibles patrones modelados mediante los PSA. Por cada una de las acciones observadas, los PSA utilizados realizaran la traslación que corresponda y calcularan su valor de probabilidad haciendo uso de la *Ecuación 3*, donde $\gamma(s_{i-1}, \sigma_i)$ representa a la probabilidad asociada en el estado s_{i-1} al símbolo σ_i (s_0 es el estado correspondiente a la secuencia vacía).

$$PSA_k(r = \sigma_1, \dots, \sigma_{|r|}) = \prod_{i=1}^{|r|} \gamma(s_{i-1,}\sigma_1) \quad \textbf{(3)}$$

Luego, el patrón modelado por aquel PSA que presente mayor valor de probabilidad será el elegido por el agente para brindar soporte al usuario. Si bien este será tomado como punto de referencia, se seguirán tomando las acciones realizadas por el usuario y evaluando los diferentes PSA conocidos con ellas, lo cual, a la larga causara que el valor de probabilidad acumulativo disminuya cada vez más, debido a que los valores que se multiplican se encuentran en el rango (0,1]. Pese a que el comportamiento de la ecuación es el esperado, el espectro de acciones contempladas es muy amplio, llegado un punto en que quizás las acciones realizadas en un comienzo por el usuario no guardan relación con las realizadas en la actualidad, por ello se deberá tener en cuenta solo un subconjunto de las mismas, las más recientes.

El problema que se está afrontando, no es un típico problema de clasificación, donde luego de observar un conjunto completo de acciones se las asocia a una "clase" (patrón). Estamos ante un problema en el cual un agente de interfaz debe ser capaz de predecir el patrón más probable luego de cada acción observada, siendo la línea que divide el diseño de un patrón con otro frecuentemente borrosa.

Para afrontar este problema se utiliza un promedio móvil exponencial que afecta a los valores de probabilidad obtenidos con la probabilidad de transición $\gamma(s_{i-1}, \sigma_i)$, para cada acción realizada por el usuario. Los promedios móviles son conocidos por ser una herramienta que permite suavizar conjuntos de datos y a su vez facilitar la detección de tendencias en los mismos. El promedio móvil exponencial (EMA por sus siglas en ingles - *Exponential Moving Average*) [7], es una estadística que permite monitorear un proceso y dar cada vez menos importancia, con el paso del tiempo, a los datos que este utiliza. Para ello utiliza factores de peso calculados de manera exponencial, disminuyendo la influencia que tendrán los datos capturados de manera temprana y otorgando mayor relevancia a los más recientes.

El factor de peso, λ, que influirá paso a paso a cada término de la ecuación, estará en el rango (0,1), y su valor determinará el envejecimiento que la aplicación de EMA represente sobre la ecuación. λ puede ser interpretado como un porcentaje, por lo cual un $\lambda = 10\%$ será equivalente a decir $\lambda = 0{,}1$. Otra alternativa, puede ser expresar λ a partir de periodos de tiempo N, donde $\lambda = 2/N + 1$.

EMA_t representa al valor de EMA para cualquier periodo de tiempo t. El valor de EMA_1 será igual al de la probabilidad a priori de la primer acción observada σ_1 y el valor de EMA para cualquier periodo de tiempo t > 1 esta dado por la *Ecuación 4*.

$$EMA_t = \lambda\, \gamma(s_{t-1,}\sigma_t) + (1-\lambda)EMA_{t-1} \quad \textbf{(4)}$$

El parámetro λ determina el nivel de influencia que tendrán los valores de probabilidad más antiguos sobre el cálculo del EMA. Un $\lambda = 1{,}0$ implica que el último valor calculado será el único que influya en el valor del EMA. Por lo tanto, mientras más grande sea el valor de λ mayor será la influencia que tendrán las probabilidades calculadas recientemente, y mientras más pequeño mayor será la influencia de aquellos valores de EMA calculados con anterioridad. Por lo general el valor de λ ronda en el rango (0,2; 0,3) [7], aunque estos valores son arbitrarios y el valor de λ debería ser calculado de manera empírica para cada caso en particular.

Luego de cada acción observada, el reconocedor de patrones calcula el valor del EMA para cada uno de los PSA y genera un ranking con los patrones que el usuario puede estar queriendo modelar en ese momento. Si el valor del EMA que se encuentra más arriba en este ranking supera a un cierto umbral de confianza τ, se realiza una predicción. También es posible especificar este valor, de manera tal, que no se tendrá en cuenta una única mnta, sino un conjunto de posibles metas que el usuario está persiguiendo en ese momento, permitiendo de este modo la realización de un análisis más exhaustivo sobre un conjunto acotado de casos.

5 Representación de los Patrones de diseño con Modelos de Markov de Orden variable

En las secciones anteriores se habló sobre los Patrones de Diseño y la manera en que los Modelos de Markov de Orden Variable permiten modelar los objetivos perseguidos por un usuario. En base a estos modelos, se presenta a continuación como modelar un patrón de diseño mediante un PSA para más tarde ser utilizado por un agente de interfaz durante su tarea de recomendación. En esta sección se muestran los pasos seguidos para la materialización del modelo, partiendo de los mecanismos otorgados por cualquier herramienta CASE.

5.1 Creación del Alfabeto

Los patrones de diseño son generalmente modelados mediante Diagramas de Clases, dado que el nivel de legibilidad que presentan estos últimos permite visualizar de manera sencilla como un conjunto de objetos se relacionan entre sí. Para su creación se utilizan herramientas CASE, las cuales dejan al alcance de la mano cada uno de los componentes que estos poseen. En este caso, clases abstractas, concretas e interfaces, y diferentes tipos de relaciones entre estos últimos (dependencias, generalizaciones, asociaciones, etc.). Dichos componentes darán origen al alfabeto utilizado para representar las cadenas de acciones necesarias para entrenar a los PST. En la Tabla 2 se presentan las acciones consideradas junto con el símbolo asignado para la creación de los distintos modelos.

Tabla 2. Acciones consideradas para la creacion de los modelos.

Acción	*Símbolo*
Interfaz Agregada	AgrInterfaz
Clase Abstracta Agregada	AgrClassAbs
Clase Agregada	AgrClass
Relación de Agregación Creada	DefAgregacion
Relación de Composición Creada	DefComposicion
Relación de Generalización Creada	DefGeneralizacion
Relación de Asociación Creada	DefAsociacion
Relación de Dependencia Creada	DefDependencia
Relación de Implementación Creada	DefImplementacion

5.2 Representación de un patrón mediante una secuencia de símbolos

Una vez que se posee el alfabeto utilizado para representar a las cadenas, se comienza con la definición de las secuencias. Para esto, se utiliza a manera de ejemplo el patrón de diseño Factory Method (Figura 1) [4].

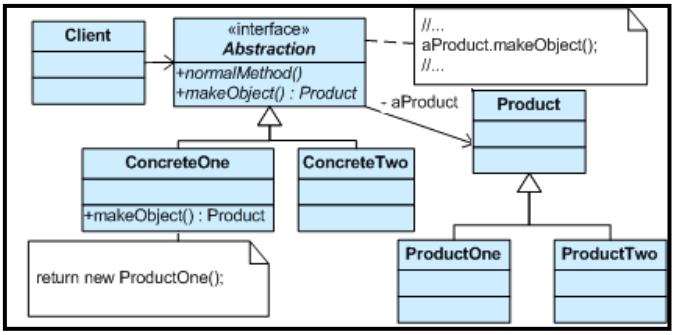

Fig. 1. Diagrama de clases correspondiente a una posible instanciación del patrón Factory Method [4].

Cada uno de los componentes que forman el diagrama, surge tras la realización de una de las acciones enumeradas en la Tabla 2. Las mismas deberán ser realizadas en un orden específico debido a que la sintaxis de los Diagramas de Clase lo exige. Por ejemplo, es imposible crear una relación de asociación entre dos clases si alguna de ellas no existe. Siguiendo esta lógica, algunas de las posibles secuencias de acciones que dará origen al diagrama son las siguientes:

1. AgrClassAbs, AgrClass, AgrClass, AgrClassAbs, AgrClass, AgrClass, DefGeneralizacion, AgrClass, DefGeneralizacion, DefGeneralizacion, DefGeneralizacion, DefAsociacion, DefAsociacion
2. AgrClassAbs, AgrClass, AgrClass, AgrClassAbs, AgrClass, DefAsociacion, DefAsociacion, DefGeneralizacion, AgrClass, AgrClass, DefGeneralizacion, DefGeneralizacion, DefGeneralizacion
3. AgrClassAbs, AgrClass, DefGeneralizacion, AgrClass, AgrClassAbs, AgrClass, AgrClass, DefGeneralizacion, AgrClass, DefGeneralizacion, DefAsociacion, DefGeneralizacion, DefAsociacion

5.3 Creación del Plan Corpus

Plan Corpus es un término utilizado para referirse a los datos de entrenamiento consistentes de una lista de objetivos y las acciones realizadas por los usuarios para alcanzarlos. En

nuestro caso particular, el objetivo es modelar un patrón de diseño y las acciones son las mencionadas en la Tabla 2. De la correctitud y completitud de este conjunto de datos dependerá la calidad de los Modelos de Markov de Orden Variables creados para la representación de los patrones.

Para la creación automática del Plan Corpus será utilizado el algoritmo de planning Ag-Ucpop [9]. Un algoritmo de planning nos permite obtener un conjunto de acciones, que al ser ejecutadas en orden permiten alcanzar un conjunto de objetivos dados [10]. En nuestro caso particular el objetivo se define como el patrón que se desea modelar y las acciones de planeamiento son las diferentes operaciones que se pueden realizar en la herramienta (esto posee una correlación directa con los símbolos de la Tabla 2). La elección del algoritmo Ag-Ucpop se debe a la simplicidad que presenta para obtener todos aquellos planes necesarios para alcanzar un objetivo en particular. La forma de utilización del algoritmo es la siguiente, mediante la definición de un estado final (un patrón de diseño), se obtienen un número de planes posibles que permiten alcanzar dicho estado y por ende definir el patrón de diseño. Cada uno de estos planes constituye un plan de orden parcial, lo cual nos permite obtener múltiples planes de orden total a partir del mismo. Cada uno de estos planes de orden total define una posible secuencia de entrenamiento.

En la Figura 2 se muestran las acciones de planning y su respectivo conjunto de pre y pos condiciones. En ellas se pueden ver las diferentes restricciones del dominio que se mencionaron anteriormente. Por ejemplo, la acción defDependencia no puede ser aplicada si antes no existe la Clase A y la Clase B, lo cual se condice con que una relación de dependencia no puede ser definida si antes no se crearon las clases correspondientes.

```
action(agregarInterfaz(I),[ ],[ ],[ interfaz(I) ] ).
action(agregarClassAbs(C),[ ],[ ],[ clase(C), tipo(C,abstracta),suelta(C) ] ).
action(agregarClass(C),[ ],[ ],[ clase(C), tipo(C,concreta),suelta(C) ] ).
action(defAgregacion(A,B),[ ],[ clase(A), clase(B) ],[ agregacion(A,B) ] ).
action(defComposicion(A,B),[ notEqual(A,B) ],[ clase(A), clase(B) ],[ composicion(A,B) ] ).
action(defGeneralizacion(A,B),[ notEqual(A,B) ],[ suelta(A), clase(A), clase(B) ],
                                [ generalizacion(A,B), not(suelta(A)) ] ).
action(defAsociacion(A,B),[ ],[ clase(A), clase(B) ],[ asociacion(A,B) ] ).
action(defDependencia(A,B),[ notEqual(A,B) ],[ clase(A), clase(B) ],[ dependencia(A,B) ] ).
action(defImplementacion(A,B),[ ],[ clase(A), interfaz(B) ],[ implementa(A,B) ] ).
```

Fig. 2. Definición de las acciones de planeamiento.

Un problema de planning consta de una 3-upla definida por el estado inicial, el estado final deseado y el conjunto de acciones que se pueden utilizar. En nuestro caso el estado inicial es vacío dado que se asume que el usuario parte desde cero, las acciones a utilizar son las especificadas en la Figura 2, con lo cual resta definir el estado final a alcanzar. El estado final, como mencionamos, se corresponde a la descripción del patrón en el lenguaje correspondiente. Por este motivo es que por cada uno de los patrones de diseño se especificara, en dicho lenguaje, un conjunto de objetos y relaciones que lo describan. A manera de ejemplo en la Figura 3 se podrá observar el Patrón de Diseño Factory Method (Figura 1) especificado en la sintaxis requerida. Una vez finalizados estos pasos, solo resta ejecutar el algoritmo y esperar los planes resultantes. Para más información respecto al funcionamiento del algoritmo referirse a [9].

clase(FabricaAbstracta), tipo(FabricaAbstracta, abst) ,clase(FabricaConcreta1),
generalizacion(FabricaConcreta1, FabricaAbstracta),clase(FabricaConcreta2),
generalizacion(FabricaConcreta2, FabricaAbstracta),clase(Cliente),
asociacion(Cliente, FabricaAbstracta),clase(ProductoAbstracto),
clase(ProductoConcreto1),generalizacion(ProductoConcreto1, ProductoAbstracto),
clase(ProductoConcreto2),generalizacion(ProductoConcreto2, ProductoAbstracto),
asociacion(FabricaAbstracta, ProductoAbstracto)

Fig. 3. Estado final asociado al Factory Method.

Como se menciono, el resultado que se obtiene de la ejecución del algoritmo consta de un conjunto de acciones que se relacionan para alcanzar el objetivo dado. En la Figura 4 se muestra un ejemplo de un posible plan resultante, en este caso el ejemplo utilizado corresponde al patrón Composite [4]. Como se puede observar, existen acciones que se encuentran en un mismo nivel, es decir que en este punto es posible obtener varios planes de orden total que cumplirán con todas las restricciones de orden.

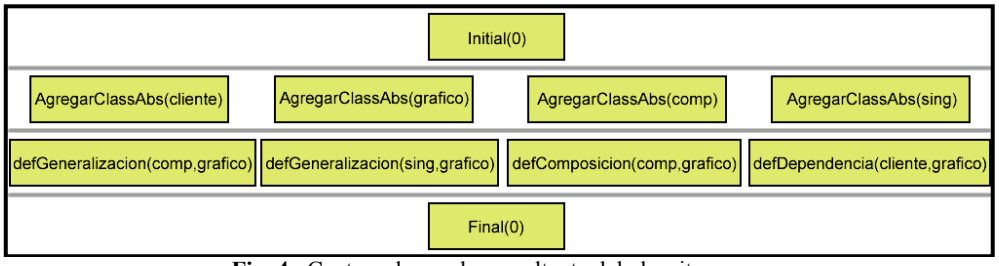

Fig. 4. Captura de un plan resultante del algoritmo.

Para el armado final de un conjunto de secuencias que permitan alcanzar un patrón dado, se extraen de la solución otorgada por el algoritmo un número de planes de orden total. Para esto se generan todos los planes de orden total correspondientes y se filtran los duplicados que se obtienen tras eliminar las variables. Dicho mecanismo se repite para cada uno de los patrones, logrando obtener de manera automática un plan corpus correcto y completo, lo cual nos asegura la calidad del modelo a crear.

5.4 Creación del PST

Una vez que se posee un conjunto de secuencias con las acciones necesarias para la creación del diagrama, se comienza con el entrenamiento del PST correspondiente al patrón al cual pertenecen las secuencias, analizando una a una las posibles acciones, y creando lazos y probabilidades de ocurrencia entre las mismas. Este proceso se repite para cada uno de los patrones de diseño.

6 Evaluaciones

En el experimento que se muestra en esta sección se utilizaran 3 métricas para evaluar los resultados obtenidos. Por un lado el *error* para un modelo Q con respecto a una secuencia $Seq = \sigma_1, \sigma_2, \ldots, \sigma_n$ se calcula como se muestra en la *Ecuación 5*, dividiendo la sumatoria del valor absoluto obtenido tras restar el valor de probabilidad calculado por el PSA para la secuencia dada en cada instante i, $Q(\sigma_i)$, con el de aquel que resulto ser el mayor en dicho instante $Q_{best}(\sigma_i)$, por la sumatoria individual de estos últimos.

$$error_q(Seq = \sigma_1, \sigma_2, \dots, \sigma_n) = \frac{\sum_{i=1}^{n} |Q(\sigma_i) - Q_{best}(\sigma_i)|}{\sum_{i=0}^{n} Q_{best}(\sigma_i)} \quad \textbf{(5)}$$

Por otro lado, se utiliza la precisión (*Ecuación 6*), que es calculada dividiendo el número de veces que el modelo buscado Q estuvo en los N mejores resultados predichos, por el número de predicciones realizadas, m. Se considera que se realiza una predicción cuando el valor de probabilidad asociado por el PSA en un instante dado, supera a un umbral de confianza τ.

$$precision_q(Seq = \sigma_1, \sigma_2, \dots, \sigma_n) = \frac{\sum_{i=1}^{n} |Q(\sigma_i) - Q_{best}(\sigma_i)|}{m} \quad \textbf{(6)}$$

La tercera y última métrica utilizada será la convergencia, la cual mide cuantas observaciones fueron necesarias para que el reconocedor dedujese el patrón buscado. Si desde el tiempo t hasta el tiempo correspondiente a la última acción observada el algoritmo predijo de forma correcta el patrón que esta modelando el usuario, la convergencia se define como se muestra en la *Ecuación 7*. El tiempo t se conoce como punto de convergencia.

$$convergence_Q(Seq = \sigma_1, \sigma_2, \dots, \sigma_n) = \frac{m - t + 1}{m} \quad \textbf{(7)}$$

$$con\ not\_best_Q(\sigma_{t-1})\ y\ best_Q(\sigma_j)\ \forall j\ t \leq j \leq n$$

$$donde\ best_Q(\sigma_i) = \begin{cases} 1\ si\ Q(\sigma_i) = Q_{best}(\sigma_i) \\ 0\ en\ otro\ caso \end{cases}$$

Las métricas presentadas tiene la finalidad de evidenciar la eficiencia con la que cuentan los Modelos de Markov al momento de predecir un patrón. De buscarse presentar métricas referentes al tiempo computacional exigido por dicho método, se debería contemplar otros métodos que tuviesen la misma finalidad que este, lo cual escapa a los objetivos de este trabajo.

6.1 Resultados Obtenidos

Para la realización del experimento se tomó un subconjunto de patrones de cada una de las categorías presentadas en [4] (3 creacionales, 4 estructurales y 4 de comportamiento). Para cada uno de ellos se codificó un ejemplo, especificado mediante un diagrama de clase que permitiese observar de la manera más clara y concisa la existencia del patrón. Mas tarde para cada uno de estos se genero tal y como se presento en la Sección 5.3, un conjunto de planes de manera computacional. Finalmente, los resultados obtenidos de este último paso sirvieron para modelar a cada uno de los patrones mediante un PST.

Una vez obtenidos los modelos necesarios para realizar las pruebas, se realizó una validación cruzada (*Leave One Out Cross Validation*) con cada una de las secuencias de acciones conocidas, siempre teniendo en cuenta cual era el patrón que se estaba modelando. Con el fin de establecer el valor más adecuado de EMA para dichas pruebas, se realizo una prueba exhaustiva del dominio utilizando valores de λ que iban desde 0.1 a 1.0, con intervalos de 0.1. Finalizada esta prueba, se llegó a la conclusión que el valor más apropiado para λ era 0.2.

En la Tabla 3, se presentan los resultados obtenidos en él experimento. Puede observarse que los valores de *precisión* obtenidos en cada modelo superan en su mayoría al 65%, con la presencia de algunos casos particulares que rondan al 30%. Estos últimos se deben a la forma en que dichos patrones son implementados, dado que presentan un alto nivel de

similitud con las estructuras básicas utilizadas en la orientación a objetos, por lo cual su formulación se hace presente en varios de los patrones con los que han sido comparados.

Table 3. Resultados experimentales.

Categoría	*Patrón*	*Precisión*	*Error*	*Convergencia*	*Longitud Prom. de las secuencias*
Creacionales	Abstract factory	0,78509964	0,02891106	69,1803	23
	Factory Method	0,95139217	0,00324753	78,8108	13
	Prototype	0,33333333	0,18535306	16,4683	7
Estructurales	Adapter	0,47037441	0,15656416	34,1058	6
	Bridge	0,80208333	0,06706767	70,3125	12
	Composite	0,26956107	0,15291051	20,9924	8
	Decorator	0,60063437	0,04938729	42,3299	12
Comportamiento	Command	0,71853189	0,04900221	60,4813	15
	Iterator	0,83858521	0,02541061	73,1404	15
	Mediator	0,67768595	0,09292792	67,7686	11
	Observer	0,71040243	0,05159191	54,9127	15
Valores promedio		0,65069853	0,07839763	53,5003	12,45

El valor de error promedio, ronda en el 7,8%. Recordar que esta métrica es independiente del número de modelos que se está considerando para realizar la predicción, dado que es calculada en base a la distancia entre el valor de probabilidad de la meta actual del usuario y de aquella que obtuvo el mayor valor de probabilidad por parte del reconocedor de planes. La convergencia promedio es del 53%, con los mismos casos particulares que la precisión donde algunos rondan en valores inferiores al 34%.

Tras observar los valores porcentuales de las pruebas realizadas, podemos concluir en que la combinación de un algoritmo de planeamiento con Modelos de Markov de orden variable, para la formulación de planes y la realización de predicción sobre el área de Patrones de Diseño, otorgan valores con un alto nivel de certeza sobre las metas buscadas por un usuario al momento de inmiscuirse en la tarea del diseñado de un software.

7 Trabajos Relacionados

La utilización de Plan Corpus generados de manera artificial, fue un tema previamente tratado por [11], quien busca formular dichas entradas a partir de la combinación entre un Algoritmo de Planeamiento modificado (SHOP2) y el Método de simulación Monte-Carlo. Las salidas generadas por este, pese a ser validas, en muchas oportunidades resultaban redundantes, situación que no se presenta en nuestro caso debido al filtrado que se realiza a los planes generados por el Ag-Ucpop. La existencia de información redundante dentro de la utilizada para entrenar a los Modelos de Markov, desencadenaría en la formulación de valores erróneos de probabilidad para las traslaciones de los PST (PSA) con que estos últimos están modelados.

En otras oportunidades se ha buscado instruir sobre la mejor forma de realizar un diagrama de clases, más precisamente sobre el cómo aplicar un patrón de diseño, utilizando un agente. Si bien dentro de los fines de este trabajo no se contempla la implementación del agente, sino la formulación de aquella información necesaria para que este realice

recomendaciones, podemos nombrar el trabajo de [3], donde se presenta un agente de interfaz para la asistencia del usuario durante la tarea de diseñado de software. En este, el medio utilizado por el agente para representar los posibles acciones a realizar por el usuario son Redes Bayesianas. Estas últimas, para cumplir con su funcionalidad, necesitan del entrenamiento a partir del conocimiento de expertos, logrando así que aquellas decisiones tomadas para aconsejar al usuario durante cada una de sus acciones sean certeras; dicho requisito, no necesariamente se presenta al utilizar los Modelos de Markov de Orden Variable alimentados por Plan Corpus generados de manera computacional. En ambas metodologías los tiempos computacionales necesarios para tomar una decisión son similares, pero esta última presenta la ventaja de poder otorgar consejos con un alto nivel de certeza sin la experiencia de un experto por detrás.

8 Conclusiones

En el presente trabajo, se propuso la aplicación de Modelos de Markov de Orden Variable para modelar las secuencias de acciones necesarias para crear un patrón de diseño. Los modelos construidos son utilizados luego para detectar el patrón que el usuario está intentado diseñar y asistirlo de manera transparente y correcta.

Otro de los aspectos tratados, es la creación del Plan Corpus necesario para entrenar a dichos modelos de una manera puramente computacional, utilizando para esto un algoritmo de planeamiento.

Como trabajos futuros se propone el estudio de todos los Patrones de Diseño definidos en el catalogo de Gamma[4], utilizando no solo diagramas de Clase para representarlos sino también Diagramas de Secuencia así como la definición de métodos y variables dentro de las clases, lo cual se espera mejorará los resultados obtenidos.

9 Bibliografía

1. Maes, P. Agents that reduce work and information overload. Communications of the ACM. 37(7):31-40. 1994.
2. Whitworth, B. Polite computing. Behaviour and Information Technology, 24(5):353-363. 2005.
3. Berdún, L., Díaz Pace, J. A., Amandi, A., and Campo, M. Assisting novice software designers by an expert designer agent. Expert Syst. Appl. 34(4): 2772-2782. 2008.
4. E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design patterns, elements of reusable object-oriented software, Addison-Wesley. 1994.
5. Rissanen, J. A Universal Data Compression System. En: IEEE Transactions Theory 29 (5): 656–664. 1983.
6. Ron, D., Singer, Y., and Tishby, N. The power of amnesia: Learning probabilistic automata with variable memory length. Machine Learning, 25(2- 3):117-149. 1996.
7. Armentano, M. G. Recognition of User Intentions with Variable-Order Markov Models. PhD thesis, Universidad Nacional del Centro de la Provincia de Buenos Aires. Argentina. 2008.
8. Hunter, J.S. The exponentially weighted moving average. Journal of Quality Technology, 18(4):203-209. 1986.
9. Luis Berdun, A. Amandi. Planning para agentes inteligentes. En: Proceedings del 7º Simposio Argentino de Inteligencia Artificial - ASAI 2005 - Rosario – Argentina. pp 12-23. 2005.
10. Weld, D.S. An Introduction to Least Commitment Planning. En: AI Magazine, 15(4):27-61. 1994.
11. Blaylock, N. and Allen, J. Generating artificial corpora for plan recognition. En: Int. Conf. on User Modeling´05, Lecture Notes in Artificial Intelligence 3538, pages 179–188. Springer. 2005.