Data Quality Evaluation

Una herramienta para evaluar la calidad de los datos en un sistema de información multi-fuente

Proyecto de grado - Marzo 2004

Integrantes:
Fabián Fajardo
Ignacio Crispino

Tutores:
Verónika Peralta
Raúl Ruggia

Universidad de la República - Facultad de Ingeniería - Instituto de Computación

RESUMEN

Cada vez existen más fuentes de información disponibles. Los usuarios necesitan acceder a la información de una manera uniforme, aunque dichas fuentes de datos, pueden ser heterogéneas e independientes y los datos pueden estar representados en diferentes formatos. Brindar a los usuarios respuestas uniformes y de alta calidad se hace difícil.

Una consulta puede tener resultados sustancialmente diferentes dependiendo de las formas de cálculo empleadas para su elaboración. Se hace necesario estudiar la calidad de los datos devueltos por cada alternativa de cálculo, para presentar al usuario la que mejor se adecua a sus necesidades.

En este documento se presenta el diseño e implementación de una plataforma para evaluar las propiedades de calidad de cada alternativa de cálculo, adaptándose a los diferentes casos de estudio. Las propiedades de calidad son calculadas por algoritmos de evaluación, que ejecutan sobre la plataforma. Nuevos algoritmos implementados pueden ser agregados de una forma sencilla.

ABSTRACT

There are many accessible sources relations. Users need to access in a uniform way to these sources, even they are heterogeneous and independents. Give users uniform answers with high quality data becomes difficult.

Queries can have different results depending on the calculation process used to solve them. It becomes necessary to study results data quality in order to give users the one that is better adapted to their needs.

This document presents the design and implementation of a platform that allow user to evaluate calculation process quality. Data quality properties can be calculated used algorithms that execute over the platform. New algorithm designed for calculate new data quality properties can be plugged in the platform in an easy way.

ÍNDICE

Apéndices

INTRODUCCIÓN

Este proyecto presenta una herramienta que permite calcular propiedades de calidad de las consultas realizadas sobre múltiples fuentes de información. En esta sección se presenta el contexto y el problema a resolver.

1.1 Contexto

Cada vez existen más fuentes de información disponibles, ya sean internas a una organización o accesibles a través de Internet. Los usuarios necesitan acceder a la información de múltiples fuentes de datos de una manera uniforme, aunque dichas fuentes, pueden ser heterogéneas e independientes y los datos pueden estar representados en diferentes formatos. Debido a la heterogeneidad de las fuentes resulta difícil evaluar la calidad de los datos para brindar a los usuarios respuestas uniformes y de alta calidad.

La calidad de los datos devueltos depende de la calidad interna de las fuentes (la coherencia, la completitud, la frescura, etc.), de la confianza sobre quién produce los datos de estas fuentes, y también de la forma de calcular los resultados.

Una consulta puede tener diferentes resultados dependiendo de las formas de cálculo empleadas para su elaboración. Las fuentes de las cuales se extrae la información, las operaciones realizadas con los datos obtenidos y las transformaciones que se aplican inciden en el resultado. Por consiguiente, la información devuelta al usuario puede ser sustancialmente diferente dependiendo de la alternativa de cálculo elegida. Se hace necesario estudiar la calidad de los datos devueltos por cada alternativa, para presentar al usuario la que mejor se adecua a sus necesidades.

La comparación de las diferentes formas de cálculo se realiza utilizando las propiedades de calidad de los resultados. Estas propiedades miden ciertos aspectos del resultado que son de importancia para los usuarios, como por ejemplo, la frescura de los datos, el tiempo de respuesta y la disponibilidad de las fuentes.

1.2 Ejemplo

Supongamos que un usuario realiza la consulta "¿Qué partidos de fútbol se disputarán por la eliminatoria para el mundial de Alemania, a qué hora se jugarán y en qué ciudad?". Esta consulta puede responderse en diferentes sitios de Internet (Conmebol, sitios de prensa locales, sitios de prensa de diferentes países sudamericanos). La información de estas fuentes puede ser distinta, alguna página puede estar desactualizada, las horas de los partidos pueden estar expresadas en la hora local de cada país, la prensa de algún país puede contener únicamente el partido que juega ese país, la información puede estar incompleta (no tener información de la ciudad donde se juega el partido), etc.

La consulta que realiza el usuario puede ser resuelta de diversas formas. Por ejemplo, obtener los partidos y la hora de la prensa local y la ciudad desde la página de la Conmebol. Otra opción es obtener información de la prensa local de cada país y luego hacer la unión de los resultados. En este caso podrían haber partidos repetidos que habría que filtrar.

La calidad del resultado dependerá de la forma de resolver la consulta. Por ejemplo, si un partido cambiase de fecha o de hora, algunas fuentes podrían no actualizar sus datos instantáneamente. Si la información es obtenida de esas fuentes, el resultado presentado estará desactualizado. Otro factor a tener en cuenta es la completitud del resultado. Si se consultara solamente sitios de la prensa local pueden obtenerse resultados incompletos. Por ejemplo, podría no contener información de los partidos que no son de relevancia para la selección local. Estos y otros factores hacen a la calidad del resultado.

Los usuarios podrían tener diferentes preferencias sobre la calidad de la información obtenida. Un usuario podría preferir la mayor completitud posible en el resultado, ya que le interesa conocer la fecha de todos los encuentros. Otros usuarios preferirán que la respuesta esté lo mas actualizada posible ya que les interesa conocer el resultado de los últimos partidos jugados.

Debido a la existencia de muchas alternativas de cálculo resulta interesante el estudio de las propiedades de calidad de las consultas, de forma que los usuarios puedan tomar la decisión de cuál se adecua mejor a sus necesidades.

1.3 Descripción del problema

Para poder elegir entre los procesos de cálculo alternativos que responden a las consultas de los usuarios, se necesita una herramienta que permita medir y comparar las propiedades de calidad de cada proceso.

Los factores que inciden en la calidad del resultado pueden ser muchos y dependen del tipo de aplicación. Por ejemplo, en una aplicación que muestra en vivo las posiciones de los autos en una carrera de fórmula 1, la realidad que se representa cambia constantemente, por lo que la frescura de los datos es muy importante. En cambio, en una aplicación para consultar los trabajos publicados por cierta organización, no lo es, ya que los datos cambian esporádicamente.

Una herramienta de evaluación de la calidad debe ser flexible y generalizable, permitiendo la selección de las propiedades de calidad más relevantes así como la incorporación de nuevas propiedades. La incorporación de nuevas propiedades puede requerir la implementación de nuevos algoritmos de evaluación. Estos algoritmos deberán poder ser agregados a la plataforma de una forma simple. A su vez, los nuevos algoritmos de evaluación podrán requerir de nueva información en las alternativas de cálculo. Por lo tanto se hace necesario permitir la representación de diferentes alternativas de cálculo para los distintos casos de estudio.

El problema que se plantea es el de construir una plataforma que permita el "test" de diferentes algoritmos y la presentación de los resultados producidos por éstos.
La herramienta debe poder representar cualquier alternativa de cálculo, permitir incorporar nuevos algoritmos, ejecutarlos y presentar los resultados al usuario. Buscamos desarrollar una herramienta que realice una representación gráfica de los datos para permitir una mejor comprensión de los resultados por parte del usuario.

1.4 Objetivos

El objetivo de este proyecto es el diseño y construcción de una plataforma que permita la ejecución de algoritmos de evaluación de calidad de datos, permitiendo la incorporación de los algoritmos a medida que vayan siendo desarrollados.

Específicamente la herramienta deberá:

- Poder representar cualquier alternativa de cálculo, independientemente de las propiedades que contenga.
- Mostrar en forma gráfica las diferentes alternativas de cálculo.
- Permitir agregar nuevos algoritmos de evaluación de propiedades de calidad.
- Ejecutar los algoritmos sobre las alternativas de cálculo y presentar sus resultados.
- Almacenar la información de los resultados en algún formato para su posterior análisis con herramientas diseñadas para ese fin.

Se realizarán prototipos de tres algoritmos de evaluación de calidad sobre ciertas propiedades específicas. Estos algoritmos se ejecutarán sobre la plataforma, lo que contribuirá a una evaluación de la plataforma en sí misma.

La documentación correspondiente al diseño y la implementación del sistema es de especial relevancia, ya que los prototipos implementados serán utilizados y extendidos en futuros trabajos del grupo CSI[1].

1.5 Metodología de trabajo

La metodología utilizada es iterativa e incremental. La realización del proyecto se separa en tres fases; en cada una de ellas se parte de un prototipo anterior, mejorándolo y agregándole nuevas funcionalidades.

En la primera fase se estudia el problema que se quiere resolver, se analiza e implementa un primer algoritmo y se realiza un prototipo de la plataforma.

La segunda fase comienza con el estudio de la plataforma y las modificaciones necesarias para poder incorporar nuevos algoritmos. Una vez modificada la plataforma, se implementa un nuevo algoritmo el cual es agregado a la misma.

La última fase consiste en la generalización de la plataforma y en la implementación de funcionalidades para mejorar la calidad del producto obtenido.

1.6 Organización del documento

El resto del documento se organiza de la siguiente manera:

En la sección 2 se definen los conceptos básicos y la terminología utilizada en este proyecto. En la sección 3 se explican los principales puntos del análisis, arquitectura, diseño e implementación de la solución. La sección 4 presenta el apego al cronograma durante el desarrollo del proyecto. La sección 6 concluye.

En el Apéndice A se encuentra un documento detallado con los requerimientos de la plataforma. El Apéndice B contiene el manual de usuario de la plataforma construida. En el Apéndice C se tiene el documento detallado de arquitectura y diseño de la plataforma. En el Apéndice D se muestran todos los archivos de definición del formato de entrada y salida de los datos. Por último, el Apéndice E contiene archivos de ejemplo para cada uno de los archivos de definición.

[1] CSI – Grupo "Concepción de Sistemas de Información" del Instituto de Computación de la Facultad de Ingeniería - http://www.fing.edu.uy/inco/grupos/csi

2 CONCEPTOS BÁSICOS

En esta sección se resumen los conceptos más importantes para comprender la plataforma y los algoritmos que fueron implementados. La sección está fuertemente basada en los conceptos descriptos en [Per03a] y en [Per04]

2.1 Sistemas de información multi-fuente

Un sistema de información multi-fuente (MSIS – Multi-Source Information System) es un sistema de información que accede a datos de diferentes fuentes de información independientes, de forma de resolver consultas de usuarios sobre dichas fuentes. [Ked99]

Un proceso de cálculo para este sistema puede ser visto como un flujo (workflow) de actividades de cálculo, posiblemente autónomas, que realizan la extracción, transformación y presentación de los datos a los usuarios [Per04]. Las diferentes actividades pueden ejecutarse en forma autónoma y asíncrona o pueden estar sincronizadas a través de eventos. Los eventos permiten a una actividad solicitar información a las actividades previas, o notificar a las actividades sucesoras cuál es la nueva información disponible.

Los flujos de actividades se representan como un grafo acíclico dirigido, donde los nodos representan las consultas, fuentes y actividades, y las aristas representan la precedencia en la utilización de información por las actividades para sus cálculos. En la Figura 2.1 se muestra una representación de un flujo de actividades. La parte inferior de la figura muestra las fuentes de datos remotas (R_i – Sources Relations) mientras que la parte superior aparecen las consultas de los usuarios (Q_i – Queries). En la parte central del diagrama se encuentran las diferentes actividades (A_i – Activities), calculadas a partir de la información de las fuentes.

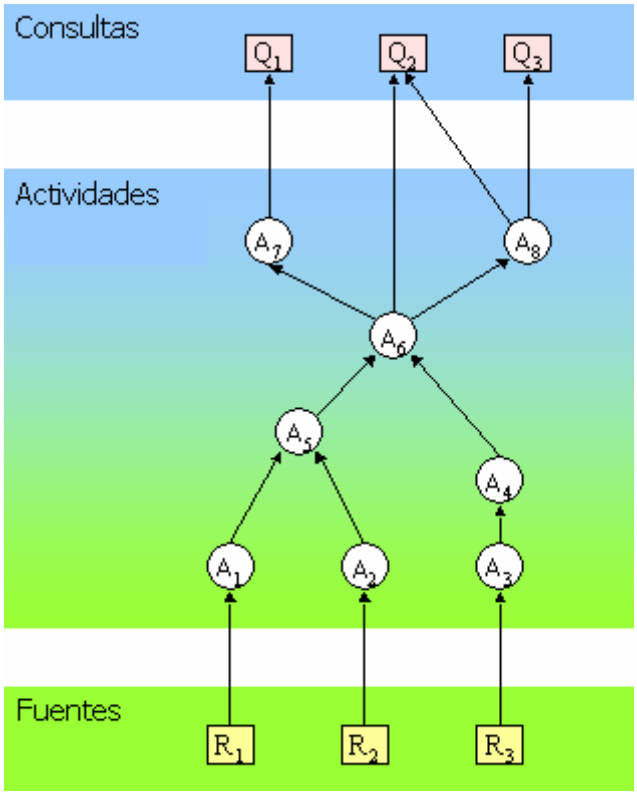


Figura 2.1 Representación del flujo de actividades [Per03]

2.2 Calculation dag

En [Per03a] se define el concepto de calculation dag (dag - directed acyclic graph) que permite representar los diferentes flujos de actividades. Un calculation dag G, es un grafo acíclico dirigido. cuyos nodos son de tres tipos:

- nodos fuentes (sin ninguna artista entrante), que representan las fuentes de datos del proceso de cálculo;
- nodos resultado o consultas (sin ninguna arista saliente), que representan los resultados del proceso de cálculo;
- nodos actividades (con aristas entrantes y salientes), que representan las diferentes operaciones utilizadas para calcular el conjunto de nodos resultados a partir de los nodos fuente.

Las aristas de G representan que un nodo es calculado a partir de otro. La información producida por el nodo de origen de la arista es utilizada por el nodo destino.

Las operaciones que realizan las actividades pueden ser por ejemplo, extracción de datos, integración o agregación. Para realizar estas operaciones toman un conjunto de datos de entrada, realizan el cálculo y eventualmente graban los resultados. A las actividades que guardan el resultado se les denomina materializadas. A las que no lo guardan se les denomina virtuales. Estas últimas pasan el resultado directamente a la actividad siguiente.

Los nodos tienen asociadas ciertas propiedades, las cuales dependen del tipo de nodo. Esto quiere decir que todas las fuentes tendrán ciertas propiedades, las actividades otras y las consultas otras diferentes. Así como los nodos tienen ciertas propiedades, también las tienen las aristas.

Por ejemplo, las actividades tienen un costo asociado que refiere al tiempo que demoran en realizar su operación. Las fuentes tienen una propiedad que indica cada cuánto tiempo son actualizados sus datos.

Cada actividad tiene una estrategia para la realización de sus operaciones. Las actividades pueden ejecutarse periódicamente, al finalizar sus actividades predecesoras o por requerimiento de ejecución de las actividades sucesoras.

Las actividades periódicas son ejecutadas cada cierto tiempo. El período de actualización es el tiempo que pasa entre dos ejecuciones consecutivas de la misma actividad.

2.3 Propiedades de calidad de los datos

En esta sección presentaremos las propiedades de calidad calculadas por los algoritmos que implementamos: frescura de los datos y tiempo de respuesta.

Frescura de los datos y factores que influyen en la frescura de los datos devueltos

La frescura se mide como el tiempo transcurrido entre el momento en que el usuario recibe los datos de respuesta a una consulta y el momento en que dichos datos fueron producidos. [Nau99]

Los usuarios imponen restricciones sobre la frescura esperada de los datos devueltos por las consultas. Este valor es obtenido directamente del usuario, en función de sus necesidades.

Cada fuente de datos tiene una frescura asociada que representa el tiempo transcurrido desde la producción de los datos. Esta frescura se puede estimar como el período de actualización de la fuente.

El costo de las actividades implica una demora en el proceso de ejecución. Por lo tanto habrá pasado más tiempo desde que se produjeron los datos hasta que son devueltos al usuario debido al costo del cálculo de las actividades.

Cuando una actividad es ejecutada periódicamente y la siguiente también, puede existir una demora entre que la primera produce los datos y la segunda los lee. Este tiempo dependerá de la coordinación entre las actividades. Si la segunda actividad comienza exactamente cuando la primera termina, no habrá demoras extras por este concepto. En cambio, si ambas actividades ejecutan con períodos diferentes o con períodos iguales pero desfasados, sí habrá una demora. A esta demora se le denomina delay entre las actividades e influye directamente en la frescura del resultado obtenido.

En la figura 2.3.1 se muestra un ejemplo (similar al que se presenta en [Per04]) de coordinación de actividades. La actividad B se ejecuta después de la actividad A y ambas actividades se ejecutan periódicamente. Las líneas punteadas indican de qué ejecución de A obtiene los datos B.

- En el caso (a), ambas actividades tienen el mismo período y B se ejecuta exactamente después de la finalización de A, por lo que el delay es 0 (las actividades están coordinadas)
- En el caso (b), ambas actividades tienen el mismo período. Como B se ejecuta una unidad de tiempo después de A, existe un delay de 1 unidad entre ambas.
- El caso (c) es una ejecución de las actividades con distinto período. Las líneas punteadas indican de qué ejecución de A obtiene los datos B. Como se observa en la figura, el tiempo que transcurre entre la finalización de A y el comienzo de B es variable. En este caso el delay se calcula como el máximo de todos los valores.

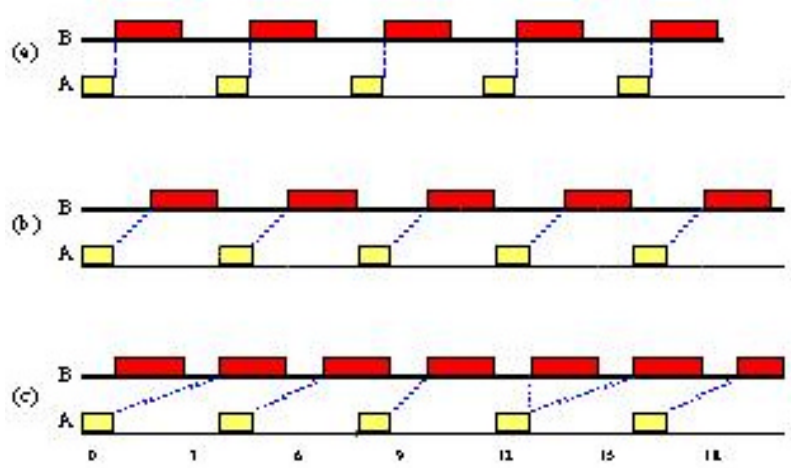


Figura 2.3.1 - Coordinación de actividades [Per04]

En función de todos estos conceptos es que se calcula la frescura de los datos devueltos. Básicamente, para el cálculo de la frescura se le asignan costos a los nodos y a las aristas del grafo. En este caso, el costo que se le asigna a las fuentes es el tiempo que transcurre entre actualizaciones. A las actividades y consultas, el costo de ejecución. A las aristas se les asigna el delay, que es calculado por el algoritmo.

A partir de estos costos, se calcula la frescura de cada nodo del grafo.

- La frescura de las fuentes es igual al tiempo que transcurre entre actualizaciones.
- Para las actividades y las consultas, se calcula como el máximo de la frescura de los nodos predecesores sumando el delay de la arista que los une y el costo del nodo.

Ejemplo: Cómo se propaga la frescura en un calculation dag.
Supongamos que tenemos un calculation dag como el que se muestra en la figura 2.3.2 En el diagrama aparecen los nodos del calculation dag con sus aristas. Los nodos R representan las fuentes. Los nodos A representan las actividades y los nodos Q representan las consultas de usuarios.
Cada cuadrado representa un nodo. En la parte superior izquierda aparece el nombre del nodo.

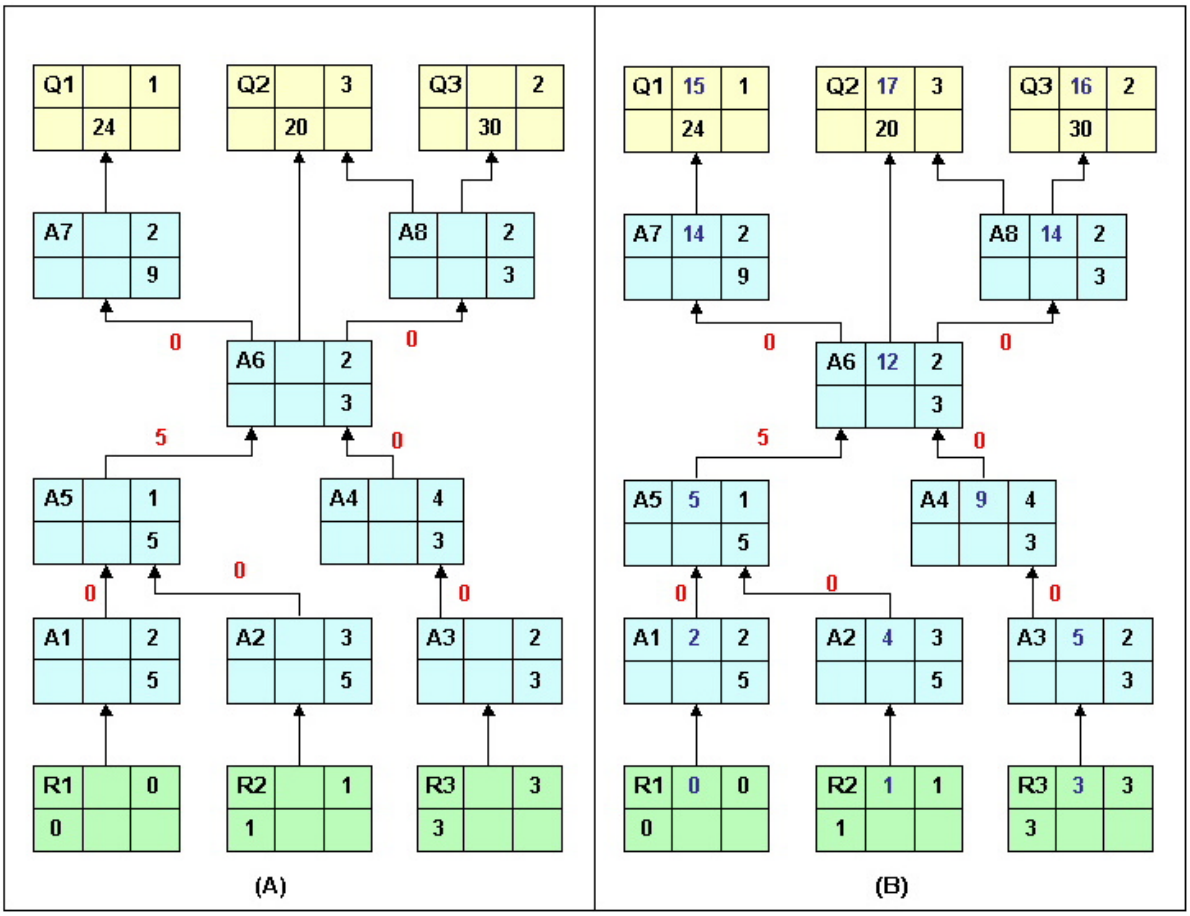


Figura 2.3.2 – Propagación de la frescura en un calculation dag

En la figura A se muestra el calculation dag con los costos de los nodos (cuadro superior derecho), los períodos de las actividades (cuadro inferior derecho), la frescura de las fuentes (cuadro inferior izquierdo), la frescura requerida por los usuario para las consultas (cuadro inferior central) y los delays entre las actividades (costo de las aristas).
En la figura B se le agregó la frescura de los datos al salir del nodo (cuadro superior central).

Veamos como se realiza el cálculo de la frescura para el nodo A6.

- Ya han sido calculadas las frescuras de los nodos que lo preceden (A4 y A5).
- Calculamos la frescura de A6 a partir de A5. Se calcula como la frescura de A5 (5) sumando el delay entre ambas actividades (5) y el costo del nodo A6 (2). Se obtiene una frescura de 12.
- Calculamos luego la frescura de A6 a partir de A4. Se calcula como la frescura de A4 (9) sumando el delay entre ambas actividades (0) y el costos del nodo A6 (2). Se obtiene una frescura de 11.
- Tomamos el máximo de los valores calculados: max{12,11} = 12.

En este ejemplo todas las actividades materializan sus datos al finalizar y se ejecutan periódicamente. En [Per04] se presenta un ejemplo para el caso general, en que no todas las actividades materializan sus resultados y algunas actividades no son periódicas, si no que se ejecutan a demanda de la actividad siguiente o al finalizar las actividades que la preceden.

El problema de asegurar la frescura

Cuando los usuarios definen las consultas, definen el valor máximo de frescura deseado. Cuando el sistema accede a las fuentes de datos, estas ya tienen cierta frescura, que se puede considerar como el tiempo transcurrido desde que la fuente de datos fue actualizada.

El problema de asegurar la frescura es el de encontrar los períodos de actualización apropiados para cada actividad de modo de coordinar el sistema de cálculo de todo el proceso, sin exceder los valores de frescura deseados para las consultas. Al cambiar los períodos de las actividades (frecuencia de actualización) se modifican los delays entre estas. Por lo tanto, varía la frescura de las actividades y por consiguiente la de las consultas.

Se considera también maximizar las frecuencias de actualización de las actividades de forma tal de reducir el costo de mantenimiento del proceso de cálculo.

Tiempo de respuesta

Como ya vimos las actividades pueden grabar sus resultados al finalizar su ejecución (materializadas) o pasar directamente el resultado a la actividad siguiente (virtuales).

Al dispararse una consulta de usuario, se necesita obtener la información que calculan las actividades que la preceden. Si las actividades son virtuales, se necesitará calcular en ese momento la información que devuelve la actividad. Si es materializada, la información ya estará calculada.

El tiempo de respuesta referencia al tiempo que pasa entre que el usuario realiza la consulta y el momento que obtiene la información.

El tiempo de respuesta de una consulta se calcula como su costo sumado al máximo tiempo de respuesta de las actividades que la preceden.

- Para las actividades materializadas el tiempo de respuesta es cero.
- Para las actividades virtuales es el máximo tiempo de respuesta de los nodos que la preceden sumando el costo de la actividad.

A continuación se define lo explicado en el párrafo anterior. TR refiere a tiempo de respuesta.

- TR(consulta) = Costo consulta + {Max(TR(n)) / n pertenece a predecesores de la consulta}
- TR(act mat) = 0
- TR(act virt) = costo act + {Max(TR(n) / n pertenece a predecesores de la consulta}

2.4 Sesión

La finalidad de la plataforma es la de permitir ejecutar algoritmos sobre diferentes alternativas de cálculo que resuelven ciertas consultas sobre fuentes de datos dadas. De esta forma habrá muchas alternativas (calculation dags) que resuelven las mismas consultas a partir de las mismas fuentes de datos.
De estas alternativas, interesa obtener ciertas propiedades, que serán calculadas a partir de ciertos algoritmos dados.

Por lo tanto durante el proceso de evaluación se fijan las consultas y las fuentes de datos. A partir de ese momento se agregan los algoritmos y las alternativas de cálculo.
Seguidamente se ejecutan los algoritmos sobre las alternativas de cálculo para obtener los valores de sus propiedades de calidad.

En base a estos elementos podemos definir una sesión como

- un conjunto de fuentes de datos,
- un conjunto de consultas,
- un conjunto de calculation dags que resuelven las consultas
- un conjunto de algoritmos que se ejecutarán para evaluar las propiedades de calidad de cada una de las consultas.

3 DISEÑO E IMPLEMENTACIÓN

En este capítulo se describe con más detalle la aplicación. Se detallan los requerimientos y se muestra el diseño de la arquitectura y los principales componentes que la integran.
Para los principales componentes se menciona su objetivo y la interacción con los demás componentes del sistema.

3.1 Requerimientos

Se desea construir un sistema para el testeo y ejecución de diferentes algoritmos de evaluación de calidad sobre calculation dags. La herramienta debe poder representar cualquier alternativa de cálculo (calculation dag), permitir incorporar nuevos algoritmos, ejecutarlos y presentar los resultados al usuario. Se pretende una herramienta que realice una representación gráfica de los datos para permitir una mejor comprensión de los resultados por parte del usuario.

La herramienta debe manejar el concepto de sesión. Debe permitir crear una sesión a partir de un conjunto de fuentes de datos (sources), consultas de usuario (queries) y algoritmos, y permitir agregar las alternativas de cálculo. Las sesiones podrán ser guardadas por el usuario. Se deberá poder abrir luego las sesiones manteniendo toda su información.

Se deberá permitir agregar nuevos algoritmos sin necesidad de recompilar la aplicación. Simplemente indicando la clase que lo implementa y un nombre para el algoritmo, este deberá ser agregado a la sesión.
Durante la ejecución de un algoritmo, la aplicación no debe quedar bloqueada a la espera de su finalización. También debe prever el caso que el algoritmo produzca un error o que su ejecución nunca termine.

Se deberá generar un reporte para comparar los resultados obtenidos. El reporte tendrá forma de tabla, en la que se pondrán en las columnas las diferentes alternativas de cálculo y en las filas las propiedades de calidad. En las celdas aparecerá para cada consulta (query) de las alternativas, el valor para la propiedad. Esta tabla deberá poder ser guardada en una base de datos para su posterior análisis con herramientas diseñadas para ese fin.

Se pretende implementar dos algoritmos para el testo de la plataforma. Estos algoritmos estarán basados en los conceptos de frescura de datos y serán especificados a partir de informes técnicos.
Los algoritmos que se implementen deberán poder ser parametrizados por el usuario. Esto es, que los nombres de los tipos de nodos y relaciones, y las propiedades que éstos tengan no estén fijas en el algoritmo, sino que sean configurables.

3.2 Objetivos de diseño

Para el diseño de la herramienta se establecieron objetivos, los cuales se pueden dividir en dos partes, los objetivos para la representación de un calculation dag y los objetivos para la aplicación en sí.

Los objetivos de la aplicación en sí que se establecieron son:

- Un cambio en la implementación de los principales componentes debe ser transparente para el resto de la aplicación.
- El manejo de Algoritmos (agregar, quitar, modificar) debe ser trivial, y sin la necesidad de recompilar la aplicación, ya que el principal objetivo de esta herramienta es el "test" de algoritmos que se irán desarrollando y agregando al conjunto de los ya existentes.
- Como no se conoce a priori el formato de los datos de entrada, la aplicación debe ser lo más independiente posible de este formato.
- Arquitectura clara y sencilla de entender para un usuario desarrollador. De forma que otros desarrolladores puedan hacer crecer esta herramienta, agregando nuevas funcionalidades.

- Ejecución de múltiples algoritmos en forma concurrente. Esto es necesario ya que algunos algoritmos pueden tomar cierto tiempo en ejecutarse y es deseable poder seguir utilizando la herramienta mientras sucede ésto.
- Escalabilidad y mantenibilidad. Facilita la realización de modificaciones e incorporaciones de nuevas funcionalidades.

Para la representación de un calculation dag, además de los objetivos anteriores, es primordial realizar una representación lo más genérica y parametrizable posible, de forma de que soporte una amplia gama de datos exigiendo sólo que se respete la definición de un calculation dag (grafo acíclico dirigido de cálculo).

3.3 Arquitectura

Para el diseño de la aplicación se optó por una arquitectura en capas, la cual facilita el cumplimiento de los objetivos descritos en el punto anterior.

Una arquitectura en capas es organizada jerárquicamente, de modo que cada capa provee servicios a las capas superiores y requiere servicios de las capas inferiores, restringiendo a su vez el uso de servicios sólo entre capas adyacentes. En esta aplicación en particular, también se cuenta con una capa transversal que provee servicios a todas las demás capas.
Las ventajas más importantes de este estilo de arquitectura son:

- Los sistemas basados en capas soportan diseños basados en niveles de abstracción crecientes. De esta forma, un problema complejo puede ser particionado como una serie de pasos o capas. Esto favorece la escalabilidad, mantenibilidad y claridad del sistema (los cuales son objetivos del diseño).
- Al igual que los tipos abstractos de datos, diferentes implementaciones de una misma capa pueden ser utilizadas indistintamente. Esto conlleva a la posibilidad de definir interfases de capas estándar, permitiendo portabilidad a los sistemas. El encapsular los componentes principales del sistema detrás de estas interfaces permite que su sustitución sea transparente para el resto de los componentes. También permite generar una capa de acceso a datos independizando la aplicación de la representación de los mismos.
- Facilita su reutilización.

Para aumentar dicha separación se optó por una comunicación entre capas por medio de interfaces, es decir, el acceso a los servicios de cada capa será estandarizado, lo que posibilita intercambiar las diferentes implementaciones de los componentes, sin que los demás sean afectados.

A su vez para el manejo de la representación gráfica se seguirá el patrón Model-View-Controller, lo que permite un desacoplamiento de los distintos conceptos y funciones. A su vez este patrón está implementado en 3 capas distintas (view, logic, model).

La Figura 3.3.1 muestra un diagrama de las capas de la arquitectura y la interacción entre ellas, así como los paquetes más relevantes que las componen.
A continuación se presenta una breve descripción de dichas capas.

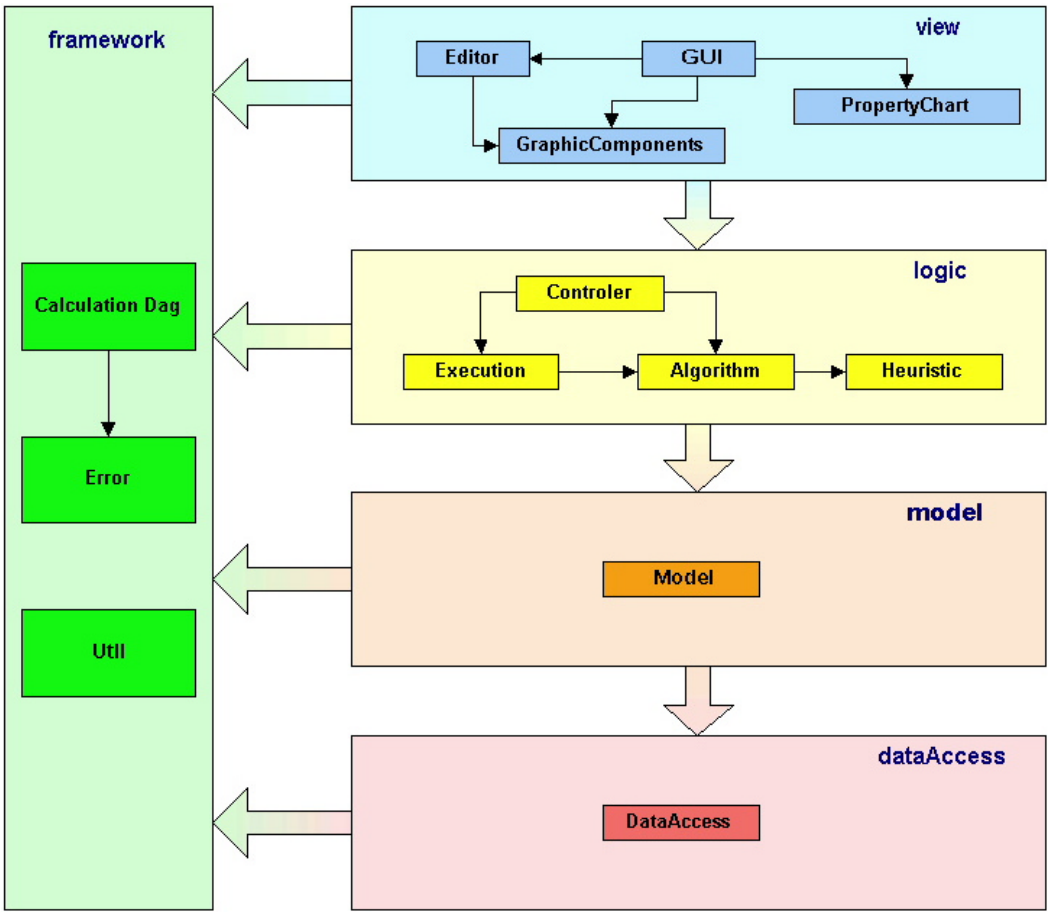


Figura 3.3.1 – Diagrama de la arquitectura

La aplicación cuenta con una capa transversal (Framework). Esta capa es accesible desde todas las otras capas, permitiendo brindar servicios básicos a todos los componentes.
El principal servicio brindado es el referente al manejo de la representación del calculation dag. Permite almacenar en memoria un calculation dag, independientemente de los diferentes nodos y tipos de nodos, relaciones y tipos de relaciones que puedan tener, así como los distintos atributos para cada uno de estos elementos.

La capa DataAccess es la encargada de acceder a los datos. Para ésto provee las funcionalidades necesarias para poder cargar un calculation dag a memoria y para poder almacenarlo en un archivo. Además para crear o guardar un calculation dag, se deben utilizar únicamente los métodos definidos en la interfaz del calculation dag.

Model lleva la información del modelo de datos que se muestra en la capa grafica. Mantiene la información de los calculation dags, las relaciones y los algoritmos.

La capa Logic es la que contiene la lógica de la aplicación. Esta capa proporciona los métodos necesarios para ejecutar algoritmos y manejar sus resultados. Posee distintos paquetes que encapsulan funcionalidades importantes para la aplicación. Por un lado Heuristic contiene implementaciones de diferentes heurísticas. Algorithm contiene los diferentes algoritmos que serán ejecutados sobre el calculation dag para calcular las propiedades de calidad, así como la interfaz que deben cumplir todos ellos. Controller es el encargado de realizar la lógica de la aplicación haciendo un puente entre el modelo de datos y su representación, y Execution es quien contiene la lógica que permite la ejecución concurrente de los algoritmos.

Por último, la capa View es la que contiene la vista gráfica que se le presenta al usuario.

3.4 Diseño de componentes

En las siguientes secciones se describe con más detalle el diseño de aquellos componentes que son importantes para la aplicación, ya sea desde un punto de vista de funcionalidad o de extensibilidad.

3.4.1 Framework

Esta capa contiene uno de los componentes más importantes del sistema, la representación del calculation dag.
Además contiene un componente para el manejo de errores y otro con clases de utilidades para que sean utiiliizadas por el resto de las capas, entre las cuales se encuentra la clase encargada de la creación de los objetos.

Creación de objetos

Como ya se ha explicado, la comunicación entre capas se realiza por medio de interfaces las cuales son utilizadas por todos los componentes para sus operaciones.

La aplicación debe permitir modificar la implementación de alguno de sus componentes sin la necesidad de modificar el resto. Para reducir el impacto de una modificación se decidió que la creación de todos los objetos fuera realizada por un único componente. Para ello se introdujo la clase Factory dentro de la cual se crean todos los objetos que constituyen la implementación de las interfaces de la comunicación entre las capas. El resto de las clases se manejan por medio de las interfaces y se limitan a solicitar una instancia de la implementación de la interfaz a la Factory, absteniéndose de crearlos. Al cambiar la implementación de uno de los componentes, basta con modificar la Factory.

Calculation dag

Este paquete es el que permite representar las alternativas de cálculo.

Veamos ahora como se diseñó internamente el calculation dag. Para ello iremos mostrando el proceso seguido para comprender mejor el diseño obtenido y las razones por las que se eligió.
Si vamos a la definición vemos que se definen 4 tipos de entidades: nodos fuentes, nodos actividades, nodos consultas y dependencias (aristas) que los unen. Como vemos, los nodos son clasificados en tres tipos, vistas, actividades y consultas. Para cada uno de estos tipos tenemos diferentes propiedades. Por ejemplo, las consultas tienen un período de actualización, las actividades tienen un costo asociado y las consultas tienen propiedades de calidad.
Las aristas también se pueden clasificar según sus propiedades. Existe un delay entre los nodos. Este delay existe solamente entre nodos actividades. No existe entre nodos fuentes y actividades, o entre actividades y consultas. Por tanto podríamos tener una clasificación para las aristas que podría ser, a groso modo, con delay o sin delay.
Generalizando lo anterior, se obtiene que un calculation dag puede ser representado como un conjunto de nodos y un conjunto de aristas, los cuales se pueden subdividir según sus propiedades.

Se debe representar cualquier calculation dag, manejando una cantidad de nodos variables así como múltiples dependencias entre ellos. Los atributos de los nodos no deben ser fijos, ya que no todos los nodos tienen que tener los mismos atributos y sus valores pueden ser de tipos de datos distintos. Lo mismo sucede con las relaciones entre ellos, se debe permitir que los tipos de datos y atributos de estas dependencias sean variables.
Esta premisa fue la guió el diseño de este componente.

La Figura 3.4.1.1 ejemplifica lo anterior. En ella se representan dos nodos genéricos y una dependencia entre ellos (Nodo2 depende de Nodo1). Tanto los nodos como la dependencia poseen una cantidad variable de atributos y tipos de datos.

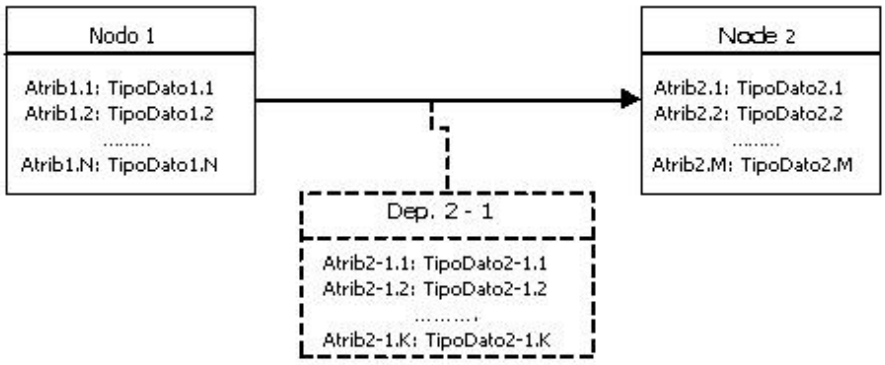


Figura 3.4.1.1 – Nodos y dependencias

El calculation dag se puede dividir básicamente en dos clases de entidades, los nodos y las dependencias entre ellos.
A su vez se pueden identificar conjuntos de entidades que representan una misma idea y poseen los mismos atributos, como ser Sources, Queries y Activities para el caso de los nodos.

La Figura 3.4.1.2 muestra la división del calculation dag en entidades y la subdivisión de cada entidad en clases o tipos.

Como se ve en la figura, se le llama "Relation" a un tipo de dependencia y se define el tipo general como Relation Type. Es importante no confundir ésta dependencia con "Source Relations" que son las fuentes de datos y que se les llamará, de aquí en más, simplemente Sources. A su vez se define Node Type como un tipo general (o clase) de nodo.

De lo anterior se desprende que un calculation dag es un conjunto de nodos y dependencias, cada uno de los cuales puede ser asociado a un tipo más general. Esto motivó que el diseño del calculation dag se hiciera de una forma descendente, partiendo de las entidades generales y bajando hacia las específicas. De esta forma, una entidad posee todos los atributos de la clase o entidad general a la que pertenece. Entonces para definir un calculation dag, primero se definen los tipos de entidades para posteriormente ir agregando las entidades asociadas a dichos tipos.

Por ejemplo, si se define un Node Type que posee N atributos, entonces todos los Nodos que pertenezcan a esta clase contendrán dichos atributos, por lo que la definición de las propiedades del nodo está dada por el Node Type al que pertenece.

En la figura 3.4.1.3 se muestra gráficamente el ejemplo anterior

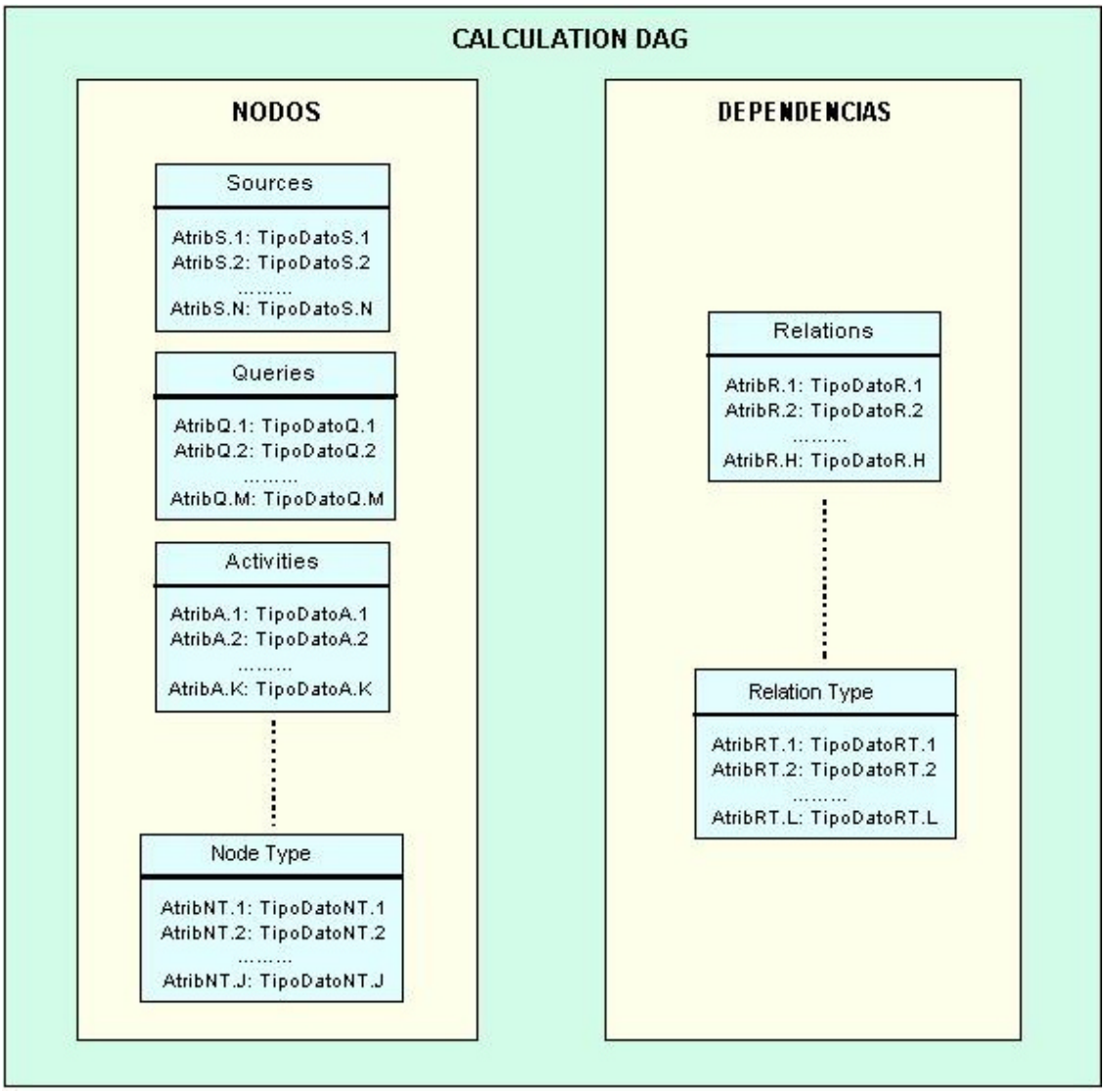


Figura 3.4.1.2 – Generalización en tipos de entidades

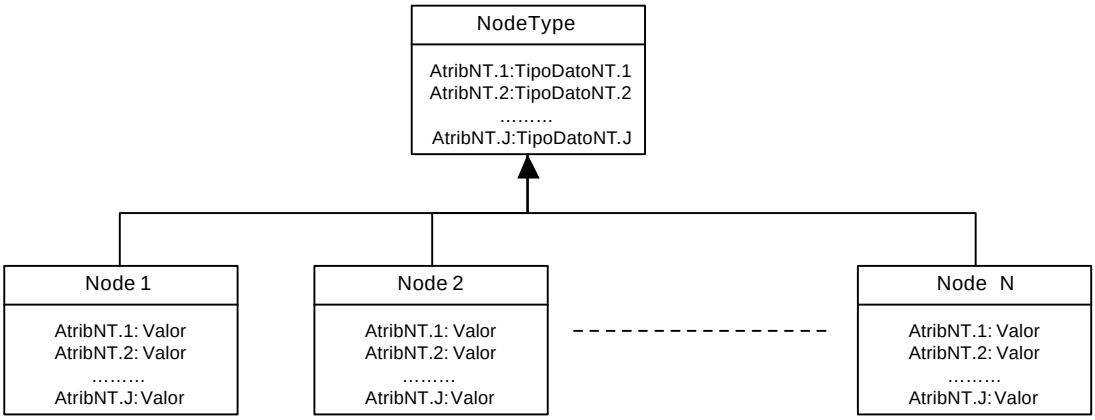


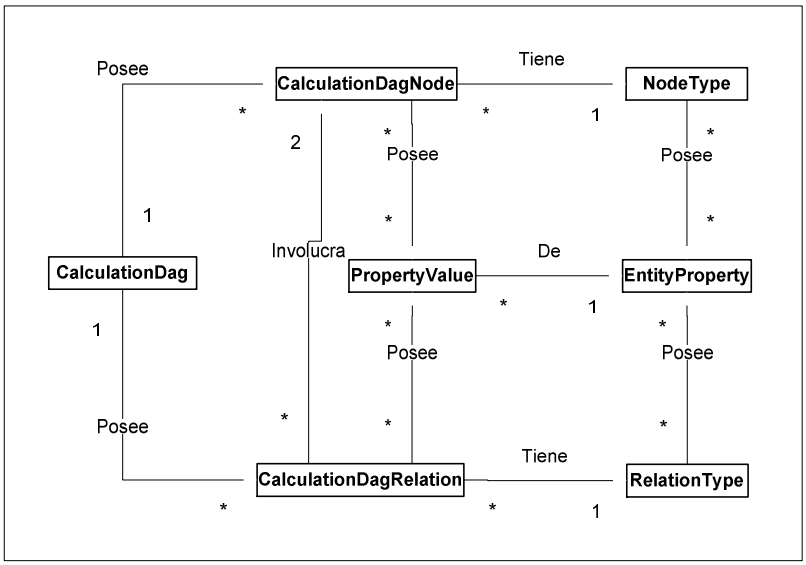
Figura 3.4.1.3 – Definición de nodos en función de su tipo

Para cumplir con todos los puntos descritos anteriormente se definió el calculation dag con los siguientes elementos:

1. Un identificador.
2. Un conjunto de Tipos de Nodos (NodeType). Estos son los nodos generales y definen los atributos con su tipo de dato para esta clase de nodo. El identificador del atributo y su tipo de dato se encapsulan en un EntityType, por lo que cada NodeType posee una colección de objetos EntityType.
3. Un conjunto de Nodos (CalculationDagNode), cada uno de ellos está asociado a un único NodeType, el cual define los atributos que posee este nodo. A su vez cada nodo posee una colección de valores para los atributos, éstos se encapsulan en PropertyValues, los cuales tienen el identificador de la propiedad y su valor.
4. Un conjunto de Tipos de Relaciones o dependencias (RelationType). Estas son las relaciones generales y definen los atributos y tipos para esta clase de dependencia. El identificador del atributo y su tipo se encapsulan en un EntityType, por lo que cada RelationType posee una colección de EntityType.
5. Un conjunto de Relaciones o dependencias (CalculationDagRelation), cada una de ellas está asociada a un único RelationType, el cual define los atributos que posee esta relación. A su vez cada dependencia posee una colección de valores para los atributos, éstos se encapsulan en PropertyValues, los cuales tienen el identificador de la propiedad y su valor.

La representación brinda los métodos necesarios para la manipulación de estos 5 elementos, es decir, altas, bajas, selección por tipo de entidad y el manejo de los valores de los atributos asociados a cada una de ellas.

La Figura 3.4.1.4 muestra la interacción entre las entidades definidas anteriormente.



No se conocen a priori todos los requerimientos de un calculation dag (tipos de nodos, propiedades, etc.), ésto lleva a que el diseño deba contemplar cualquier tipo de entidad con un conjunto arbitrario de propiedades de distintos tipos. Si bien con el diseño realizado se contempla un universo extenso de caculation dags, se prevé el caso en que se deba cambiar la implementación del mismo.
Por este motivo, la comunicación entre el Calculation dag y los demás objetos, se da a través de una interfaz ICalculationDag, la cual intercambia con las demás clases objetos String, es decir, los métodos de dicha interfaz permiten realizar todas las operaciones necesarias sobre el Calculation dag pero utilizando Strings como parámetros de entrada y de salida. Esto lleva a que en ningún

momento se accede a los objetos internos de la representación, si no que todas las operaciones son realizadas sobre el CalcultionDag directamente, haciendo transparente el cambio de la representación del mismo.

Ejemplificando lo anterior, para modificar cierta propiedad de un nodo, una opción es pedir al Calculation dag el nodo deseado y modificar sus propiedades. Si se hiciera de esta forma, se tendría una dependencia a los objetos internos del calculation dag, que haría que algún cambio sobre esta clase repercutiera en clases externas a la representación.
Utilizando los métodos de la interfaz, la modificación de una propiedad de un nodo se realiza mediante una petición al calculation dag de que se modifique la propiedad X del nodo Y (siendo X e Y identificadores de tipo String). De esta forma, es posible modificar la representación interna, sin tener que modificar las clases que utilizan los diferentes métodos sobre el calculation dag, siempre y cuando la interfaz se mantenga.
Una tercera opción es definir una interfaz para cada objeto que pudiera ser usado fuera de este paquete. Esta opción requeriría manejar una gran cantidad de interfases de objetos internos que podrían no tener sentido en otra implementación de este paquete, dificultando el diseño y la implementación de una solución totalmente diferente en caso de ser necesario.

Los valores de las propiedades de los nodos y relaciones son de tipo Object de java, de forma de poder almacenar cualquier tipo de datos. Lo único que se exige es que tenga implementada la función toString e implementado el constructor a partir de un String, ya que los datos son salvados de esta forma.

La aplicación debe permitir ejecutar un algoritmo sobre un calculation dag, obtener el resultado y mantener el calculation dag original. Para ello, este paquete cuenta con una operación que permite duplicar un calculation dag.
El calculation dag internamente está representado como un grafo dirigido y sin ciclos. Los nodos del grafo, serán los nodos del calculation dag y las aristas serán las relaciones o dependencias. La información de los nodos y de las relaciones queda contenida dentro del grafo.
A su vez, la implementación se ocupa de mantener su integridad, es decir, al agregar una relación se verifica que ésta no genere un ciclo en el grafo.

3.4.2 View

En esta capa se encuentra la interfaz gráfica que se le presenta al usuario. Dicha interfaz debe mostrar la representación de un calculation dag, la ejecución de un algoritmo y los resultados de dicha ejecución, también debe permitir comparar los datos obtenidos por medio de los distintos algoritmos.

La vista es parte del patrón Model-View-Controller, por lo que todos los datos que se muestran están almacenados en el modelo y pueden ser solicitados a través del Controller, el cual se encuentra en la capa inferior.

Las funcionalidades que brinda esta capa están separadas en paquetes, para hacer más clara su implementación.

GUI

Se tiene un paquete GUI (Graphic user inteface). Este paquete contiene la ventana principal y en ella, todos los demás componentes. Su implementación se basa en el paquete de componentes gráficos de java, swing.
Esta ventana cuenta con una referencia al paquete Controller de la capa lógica, el que le permite interactuar con el modelo de datos que está representando.

GraphicComponents

Otro paquete es el encargado de representar gráficamente los calculation dags. Este paquete es el GraphicComponents. Al diseñar la representación gráfica de los calculation dag se tenía como

objetivo que fuera amigable y en lo posible modificable en tiempo de ejecución, permitiendo que el usuario pudiera darle el orden adecuado al grafo. Es por esta razón que se eligió utilizar JavaBeans, los cuales permiten cambiar sus propiedades en tiempo de ejecución, principalmente su posición. Cada elemento del calculation dag es asociado a un componente gráfico el que se encarga de gestionar la representación gráfica del mismo (identificador, propiedades, etc.).
En una ventana interna de la ventana principal del paquete GUI, se muestra la representación gráfica de los calculation dags generada a partir de los componentes gráficos.

Property Chart

PropertyChart es el paquete que se encarga de la creación y gestión de la tabla comparativa de las propiedades de calidad.

La tabla se arma en función del cruzamiento de calculation dags y algoritmos. La información de qué calculation dags y qué algoritmos se cruzan se obtiene del usuario. El paquete implementa un wizard para realizar la interacción con el usuario.

Como la información que se muestra en esta tabla es la misma que se maneja en la ventana principal, este paquete utiliza el mismo Controller que usa la ventana principal. Esto le permite acceder al modelo de datos y ejecutar algoritmos sobre los calculation dags de la sesión. De esta forma, y debido al modelo de eventos (que se explica en la siguiente sección), las modificaciones realizadas por este paquete en el modelo, repercuten en todas las vistas de los datos.
Al armar la tabla solicita los datos de los cruzamientos al Controller. Si alguno no está en el modelo de datos, invoca sobre el Controller la ejecución del algoritmo sobre el calculation dag.

Si bien este paquete no guarda la tabla en el modelo, ya que es una vista de los mismos datos existentes, brinda la opción al usuario de guardar los resultados de los cruzamientos en una base de datos para un estudio posterior más detallado. Este componente invoca la operación sobre el Controller, la cual finaliza en el DataAccess.

Editor

El último paquete importante de esta capa es Editor, el cual contiene la implementación del editor de calculation dags. Dicho paquete permite la creación en tiempo de ejecución de un calculation dag.

La funcionalidad que debe permitir es la de crear calculation dags y permitir la visualización de éste durante el proceso.

Para la visualización del calculation dag se utilizan las mismas clases que utiliza la ventana principal, el componente CalculationDagPanel del paquete GraphicComponents.

La edición se realiza utilizando un JMenu, en el que se agregan todos los componentes del calculation dag, permitiendo el alta, baja y modificación de cada uno de ellos.

Este componente no debe permitir la modificación de los datos intrínsecos a la sesión, pues alteraría el concepto que ésta representa.

La vista solicita la edición del calculation dag pasándole dos parámetros, ambos calculation dags. El primer parámetro indica al editor cuáles son los datos intrínsecos de la sesión (tipos de nodos, tipos de relaciones y nodos). El segundo parámetro es el calculation dag base para comenzar la edición.
Esto permite crear un calculation dag desde cero (a partir de las Sources y las Queries) o utilizando uno ya existente en el modelo como punto de partida. Esta funcionalidad brinda al usuario la posibilidad de realizar fácilmente pequeñas modificaciones sobre los calculation dag y analizar el impacto que provocan en el resultado.

3.4.3 Logic

Esta capa encapsula toda la lógica de la aplicación, es la encargada de gestionar la ejecución de algoritmos, de realizar el enlace entre la vista y el modelo de datos y de crear los objetos que implementan las interfaces.

Controller

Esta capa también posee la implementación del Controller de la aplicación.
Este paquete brinda servicios a la capa gráfica, los cuales son utilizados para las diferentes funcionalidades que percibe el usuario. Para ésto utiliza los servicios de la capa Model, la cual posee el modelo de datos con toda la información de la sesión, es decir, el Controller hace de puente entre el modelo de datos y su representación gráfica. Dicha información está contenida en un objeto del tipo sesión (de la capa Model).

También posee la lógica necesaria para ejecutar los algoritmos. Para ello cuenta con la información de los algoritmos que fueron agregados a la sesión, básicamente el nombre y la clase que lo implementa. Todos estos algoritmos implementan la interfaz IAlgorithm.
Cuando se le solicita la ejecución de un algoritmo, genera una instancia de la clase asociada al identificador del algoritmo e invoca al método execute, el cual es parte de la interfaz.
Como los objetos son creados dinámicamente a partir del identificador de la clase, se permite agregar nuevos algoritmos en tiempo de ejecución, sin la necesidad de recompilar la aplicación.

Los algoritmos implementados como casos de estudio son realizados sobre los tipos más generales posibles, de forma de facilitar su reutilización. Para esto se parametrizaron en función de las propiedades, y los identificadores de las mismas se deben adjuntar en un archivo de propiedades. Esto permite ejecutar el mismo algoritmo sobre calculation dag que difieren en los identificadores de sus propiedades.
Para cada algoritmo implementado se debe proveer de un archivo de configuración en el que se indique los tipos de nodos, los tipos de relaciones y los atributos de los nodos y relaciones que utiliza el algoritmo para sus cálculos. Cada calculation dag al que se le quiera correr el algoritmo deberá estar creado en base a los identificadores del archivo de propiedades, en caso contrario se indicará un error. Esto se verá con más detalle en la sección 4.5 donde se presentan los casos de estudio.

Ejecución de algoritmos, Excecution

La ejecución de los algoritmos se realiza en una hebra separada del programa, permitiendo así, que múltiples algoritmos ejecuten concurrentemente y a su vez que el usuario pueda continuar usando la aplicación mientras éstos finalizan.
Para que el uso de esta funcionalidad sea controlado se creó una clase Executioner, la cual se encarga de la creación y gestión de las hebras. En la implementación actual, esta clase provee sólo métodos para solicitar la ejecución de un algoritmo o su cancelación. Si esta herramienta se quisiera portar a un ambiente Web, alcanzaría con cambiar la implementación de esta clase de forma que funcione como un scheduler, es decir, agregando una cola de pedidos y acotando la cantidad de hebras a utilizar.

Esquema de eventos

Este paquete posee toda la implementación del esquema de eventos que utiliza la aplicación, el cual se detalla a continuación.
Como ya se ha mencionado cada ejecución de un algoritmo se realiza en una hebra distinta, cuando dicha ejecución finaliza dispara un evento, de forma que todos los componentes interesados sean informados que la ejecución finalizó y en qué estado. A su vez en este evento viaja toda la información relacionada con la ejecución, como ser, una referencia al calculation dag resultante, el identificador del calculation dag original, el identificador del algoritmo ejecutado, el estado de finalización (OK o ERROR) y un mensaje utilizado generalmente para el caso de error.

La forma en que una clase se declara "interesada" en recibir este evento, es por medio de la implementación de la interfaz IExecutionListener, la cual posee un método que es invocado cada vez que una ejecución finaliza. Una vez implementada dicha interfaz se puede agregar como "listener" de este evento. Es responsabilidad de la clase identificar si dicho evento es de su interés utilizando la información contenida en el mismo.
El tercer actor en este esquema es el ExecutionObserver, esta clase es la encargada de mantener la lista de Listeners de los eventos. Este componente es un "Singleton", es decir, existe una única instancia del mismo. A su vez es un objeto estático, de forma que es accesible sin necesidad de tener que crearlo, alcanza sólo con pedir una instancia de él, que en realidad es única.

La Figura 3.4.3.1 muestra el esquema de eventos.

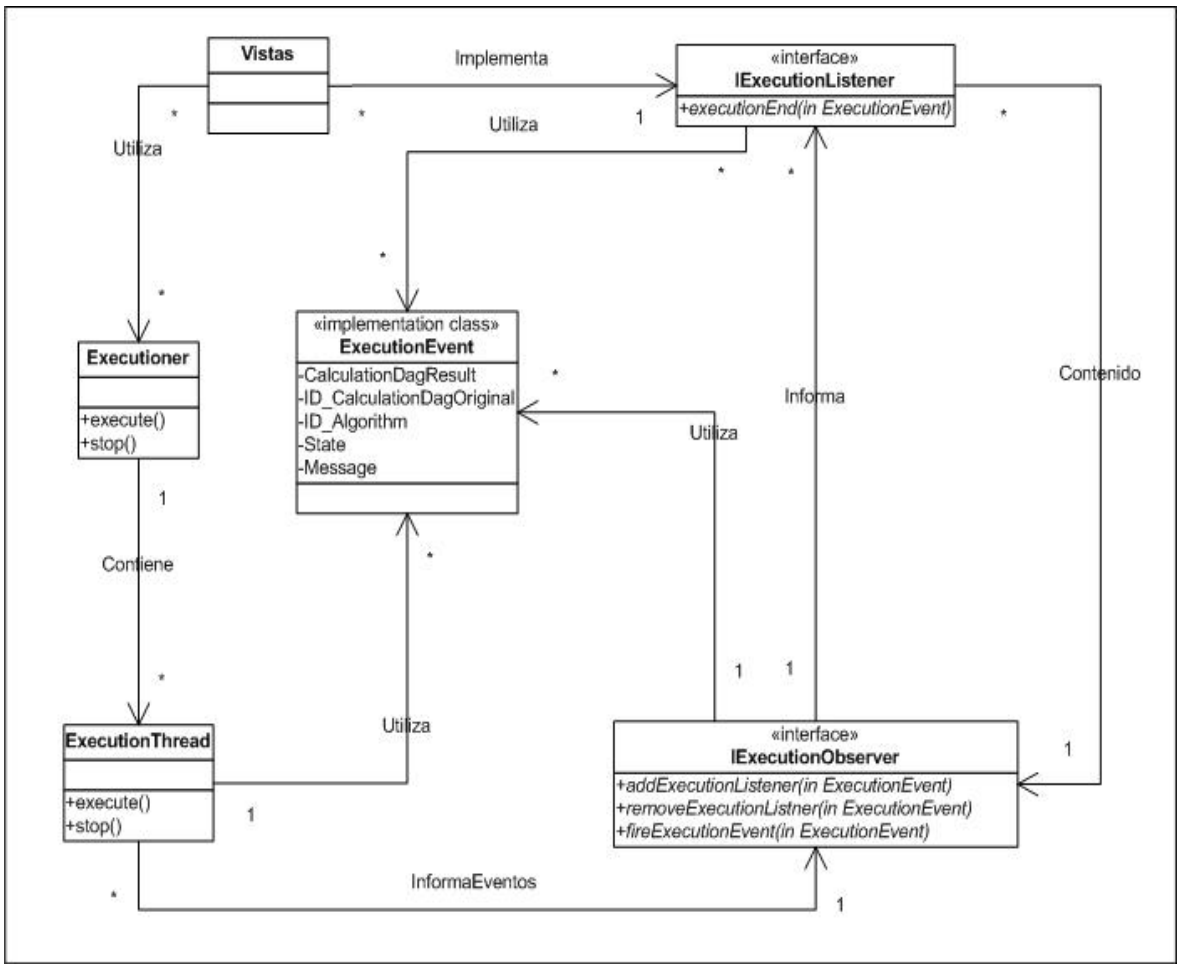


Figura 3.4.3.1 – Esquema de eventos

3.4.4 Model

Esta capa es el tercer integrante del esquema Model-View-Controller. Es la que contiene el modelo de datos donde se lleva la información necesaria para la vista e interactúa con el Controller.

El principal concepto que maneja esta capa es el de Sesión, el cual fue descrito anteriormente en este documento.

La sesión posee los calculation dag resultantes de la ejecución de los diferentes algoritmos, así como la relación entre ellos, dicha información es utilizada por la interfaz gráfica para representar una estructura arbórea, donde los nodos raíces son los calculation dag originales y las hojas son los resultados de la ejecución de algoritmos sobre la raíz.

Para esto se cuenta con una interfaz INodo la cual permite guardar información del calculation dag actual, el calculation dag original y el algoritmo aplicado.

Para el manejo de la sesión se cuenta con una interfaz, ISesion, a través de la cual se pueden realizar las operaciones sobre la sesión. Es importante destacar que se maneja una única sesión por instancia del programa.
Para utilizar una sesión se debe crear y llamar al método INIT antes de realizar cualquier operación, ya que a través de este método la sesión recibe el calculation dag que se utilizará como modelo, el cual posee las fuentes y consultas cargadas al crear la sesión.
Cuando se carga un conjunto de actividades, las mismas son agregadas al calculation dag modelo, previa duplicación del mismo.

A su vez esta capa sirve de enlace entre la lógica y el acceso a datos, proveyendo de métodos para que sean invocados por la capa lógica (guardar y cargar) y utilizando los métodos necesarios de la capa de acceso a datos.

3.4.5 DataAccess

Esta capa es la encargada de manejar el acceso a los datos. Se encarga de persistir y recuperar los calculation dags, las sesiones y los algoritmos, también se encarga de la conexión con la base de datos donde se almacena la tabla comparativa.

Como el formato de entrada puede variar, es necesario tener la posibilidad de adaptarse a diferentes formatos. Por esto se definen dos clases abstractas, una para realizar la lectura y otra para realizar la escritura. Estas clases son: CalculationDagReader y CalculationDagWriter, para leer y escribir respectivamente. Para cada formato de entrada se deberá implementar dos clases que hereden de las anteriores e implementen la lectura y escritura para el formato deseado. Esto permite que en la misma aplicación se puedan leer diferentes archivos de entrada, con diferentes formatos, dejando la posibilidad que la aplicación (a nivel de la vista) elija cuál se deberá usar en una instancia dada del programa.

A efectos de esta aplicación se tiene una implementación de las clases anteriores, la cual utiliza XML Schemas para definir el formato de los distintos tipos de archivos de almacenamiento.
Se eligió utilizar XML ya que proporcionaba portabilidad y permitía realizar una abstracción de las fuentes de datos reales.

La implementación de las clases se realiza mediante el uso de Castor (http://castor.exolab.org). Castor es una herramienta que permite generar (dado un esquema que respeta determinadas reglas), las clases necesarias para leer y escribir archivos XML acordes a un esquema.

Como los valores de las distintas propiedades del calculation dag son almacenados como un String, es necesario que todos los tipos que se manejen para dichos valores se puedan construir y guardar a partir de un String.

Para guardar los datos de la tabla comparativa se realiza una conexión con una base de datos, los parámetros de dicha conexión (como ser el driver, usuario, contraseña) son obtenidos del archivo de propiedades de la aplicación.

3.5 Interacción entre capas

Como ya se ha descrito la interacción entre capas se realiza a través de interfaces, de forma de facilitar el mantenimiento de la aplicación. A su vez existe una única clase, Factory, la cual crea las instancias de las implementaciones de las interfaces.
Otro punto a destacar, es que la comunicación entre las capas superiores e inferiores se realiza por medio de invocación de métodos, la comunicación inversa es por medio de eventos. La única excepción es la capa framework a la cual acceden todas las otras capas, pero esta capa es autónoma, no utiliza ningún componente de las otras.

En la Figura 3.5.1.1 se describe la secuencia para la ejecución de un algoritmo.

- El usuario solicita al sistema la ejecución de un algoritmo sobre el calculation dag seleccionado actualmente.
- Se crea una ventana para la representación del resultado.
- La ventana solicita la ejecución del algoritmo a la capa lógica.
- La capa lógica (en el componente Excecutioner) crea una nueva hebra en la cual se ejecuta el algoritmo.
- Una vez finalizada la ejecución, el ExecutionObserver es informado mediante un evento. Dicho componente se encarga de avisar a las vistas de la finalización del algoritmo, enviando a su vez el resultado obtenido.
- Se dibuja el resultado en la ventana creada al comienzo.

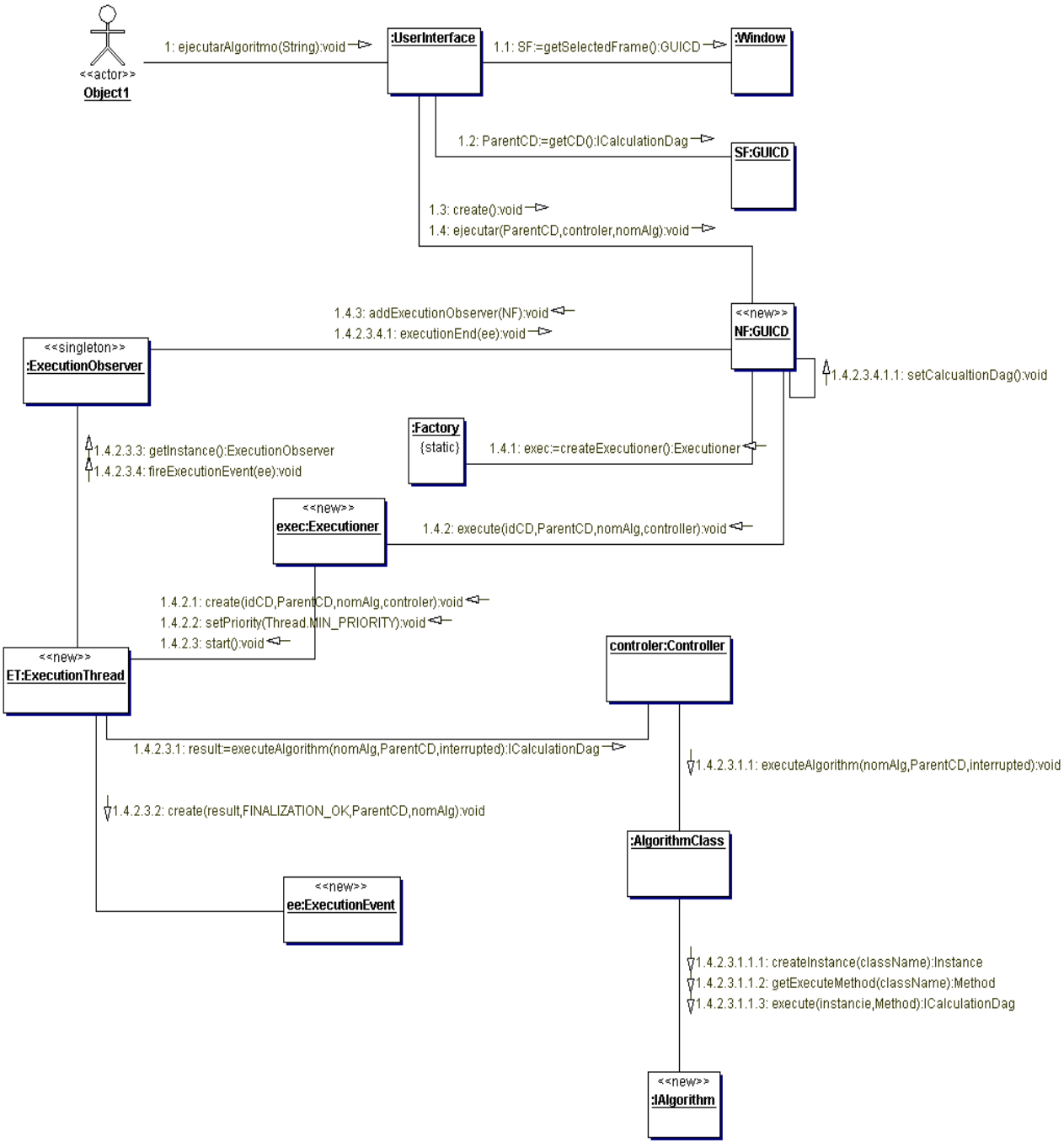


Figura 3.5.1.1 – Ejecución de un algoritmo

3.6 Algoritmos de "test" de la plataforma

Esta sección tiene como objetivo presentar la arquitectura utilizada para la definición de algoritmos, mencionar la implementación de los diferentes algoritmos desarrollados y las generalidades de los mismos.

La implementación de los algoritmos está basada en [Per03a] y [Per03b].

3.6.1 Diseño

Es necesario que la plataforma pueda incorporar nuevos algoritmo en tiempo de ejecución. Para ello es necesario definir una interfaz o una clase abstracta, para que los diferentes algoritmos puedan implementar o heredar de ella, dependiendo del caso.

Se optó por una interfaz, ya que era suficiente para la definición de un algoritmo y resulta menos restrictivo, ya que un algoritmo implementado podrá heredar de cualquier otra clase.

La interfaz diseñada cuenta únicamente con dos métodos, uno que permite a la plataforma invocar la ejecución de un algoritmo sobre un calculation dag (método execute), y otro que permite, también a la plataforma, consultar cuáles son las propiedades de calidad que el algoritmo calcula.

Método execute

El método execute es el que permite realizar la ejecución del algoritmo sobre un calculation dag. Recibe tres parámetros, el calculation dag sobre el que tiene que ejecutar, un objeto de propiedades y un objeto para indicarle al algoritmo si debe detenerse.

En la definición de esta clase se optó por hacer que la ejecución reciba un objeto de propiedades (que es levantado a partir de un archivo de propiedades). Este objeto es el que permite al algoritmo recibir de la plataforma, la parametrización necesaria. En este archivo de propiedades se pueden configurar el nombre de los atributos de nodos y relaciones que utiliza el algoritmo, el nombre de los atributos que el algoritmo devuelve, la identificación de los tipos de nodos y tipos de relaciones, o cualquier otra información que sea relevante para el algoritmo.

El tercer parámetro permite a la plataforma solicitar al algoritmo que se detenga. Consiste en un objeto con una propiedad de tipo booleano. La plataforma otorga el valor true a la propiedad cuando quiera avisar al algoritmo que debe detenerse. La implementación del algoritmo puede hacer caso omiso de esta petición u obedecerla en la medida que le sea posible.

Método getQualityProperties

Este método, es el utilizado al crear la tabla comparativa para identificar las propiedades de calidad que cada algoritmo calcula. Este método recibe un único parámetro, un objeto con las propiedades del algoritmo. Estas propiedades son las mismas que recibe el método execute. De esta forma se permite que los nombres de los atributos sean configurados desde fuera de la clase, sin necesidad de modificar el algoritmo cuando se quiere cambiar el nombre de alguna de las propiedades.

3.6.2 Algoritmos implementados

Los algoritmos implementados se basan en el cálculo de la frescura de los datos devueltos por las consultas y en el tiempo de respuesta de las consultas.

Cálculo de las frecuencias de actualización para satisfacer la frescura de los datos

El primer algoritmo estudiado fue un algoritmo para asegurar la frescura de los datos de las consultas para los requerimientos de los usuarios. El estudio y la implementación se hicieron a partir de [Per03a].
En el documento se proponían dos soluciones, una basada en un backtracking y la otra en una heurística. Se implementaron ambas propuestas. Para el caso de la heurística fue necesario realizar un estudio del problema, llevando la definición de la solución a más bajo nivel para realizar la implementación.

El problema que este algoritmo resuelve es el de hallar los períodos de actualización de las diferentes actividades, de forma de coordinar la ejecución del sistema en su conjunto y asegurar el cumplimiento de las restricciones de los usuarios.

La implementación mediante backtracking consigue la mejor solución; recorre todas las posibles soluciones, evaluando cuál de ellas es mejor. La solución mejor es la que cumple con todas las restricciones de los usuarios y tiene el costo de mantenimiento mínimo. La utilización de este algoritmo es factible cuando los procesos de cálculo son relativamente pequeños (del orden de 10 actividades). Al crecer la cantidad de actividades el tiempo en que demora la ejecución es demasiado alto.

La implementación utilizando la heurística, busca encontrar una solución que satisfaga las necesidades de los usuarios, pero puede no ser la óptima. Se basa en ir fijando los períodos de las actividades, de a una, para garantizar que se satisfagan los requerimientos de los usuarios. El algoritmo encuentra una solución mucho más rápidamente que la implementación utilizando backtracking. La contrapartida que tiene este algoritmo es que las actividades quedan fuertemente acopladas. Si el sistema sufriera alguna variación o alteración, y una de las actividades tomara más tiempo para su resolución, todo el sistema podría atrasarse (en lo que respecta a la frescura de datos) y no cumplir con los requerimientos de los usuarios.

Representación general de un sistema de mediación

El segundo algoritmo es una generalización del primero. Se implementó basado en [Per03b]

Para el primer caso se considera que todos los nodos tienen una estrategia de ejecución periódica. En este caso, los nodos pueden tener tres tipos de ejecución diferentes: periódicos (igual que el primer caso), pulled (la ejecución de la actividad se realiza cuando algún nodo sucesor solicita información) o pushed (se ejecuta cuando todos los nodos que lo preceden tienen nueva información).

La implementación de este algoritmo se realizó utilizando backtracking, que es la solución que se describe en el artículo.

La finalidad de este algoritmo es la misma que la del anterior, encontrar los períodos de actualización de las actividades que son periódicas, de forma de asegurar los requerimientos de los usuarios y de minimizar el costo total de mantenimiento.

En este algoritmo se incorpora un nuevo concepto. Como no todas las actividades materializan sus resultados, cuando el usuario realiza una consulta será necesario calcular algunos de los datos de las actividades. Esto produce una demora entre que el usuario realiza la consulta y obtiene el resultado por parte del sistema.
En el algoritmo anterior, todas las actividades estaban materializadas, por lo tanto el tiempo de respuesta era igual al tiempo de cálculo de la consulta. En este caso, el tiempo de respuesta será el camino más largo por el grafo, recorriendo únicamente aquellas actividades que no materializan sus datos.

Generalidades

Para todos los algoritmos implementados se parametrizaron los nombres de las actividades, los nombres de las relaciones y los nombres de las propiedades. Se realizó utilizando el parámetro de propiedades que recibe el método execute.

Se implementó también, para el caso de los algoritmos implementados con backtracking, la funcionalidad de detener el algoritmo.

4 EJECUCIÓN DEL PROYECTO

En esta sección se presenta el cronograma propuesto al comienzo del proyecto y el realizado realmente en el transcurso del mismo.

4.1 Cronograma propuesto

La realización del proyecto se puede separar en 6 fases que se detallan a continuación:

Tarea	Descripción	Duración
Análisis	Estudio del mecanismo a resolver. Modelización de alto nivel. Presentación de los resultados de análisis y problemas a resolver	1 mes
Primer Prototipo	Prototipación de la plataforma con un primer algoritmo de evaluación de calidad Presentación del prototipo. Documentación asociada	2 meses
Testeo del prototipo	Elaboración de un caso de estudio y datos de prueba para testear el algoritmo implementado. Presentación de los resultados de la investigación y problemas por resolver	1 mes
Análisis y Diseño detallados	Estudio de la incorporación de nuevos algoritmos de evaluación a la plataforma. Análisis de interfases. Diseño de la solución. Documento de diseño.	1,5 meses
Implementación	Incorporación de otro algoritmo de evaluación a la plataforma. Desarrollo de interfases y entrada/salida de datos. Testeo	1,5 meses
Terminación	Puesta a punto, afinamiento de detalles Revisión y organización del informe. Manuales de usuario y de instalación	1 mes

4.2 Cronograma ejecutado

A grandes rasgos se puede decir que se cumplió con el cronograma, aunque algunas tareas llevaron más tiempo y otras menos tiempo del planificado. En esta sección se describe como se realizó el proceso.

Análisis

La tarea de análisis se realizo según el cronograma. Durante esta tarea la dedicación horaria fue más baja que en el resto del proyecto.

Primer Prototipo

El objetivo planteado por el cronograma era el de tener un prototipo que después se modificara. En lo referente a la representación de los calculation dags, se fijó como objetivo, al comienzo de esta fase, hacerlo lo más parametrizable posible con el fin de que no sea modificado en instancias posteriores. La plataforma, en cambio, se diseñó sabiendo que luego iba a ser modificada, pero tratando que la base de la plataforma permaneciera en la versión final de la aplicación. Esto provocó que se le invirtiera más tiempo del planificado al diseño e implementación de la solución.
Se realizó también una primera propuesta de formato de entrada/salida de datos. En el prototipo se implementó el paquete de acceso a datos utilizando el formato de entrada/salida definido.
Al finalizar la etapa se realizó la documentación correspondiente a la arquitectura y diseño de la solución.
Se implementó además un primer algoritmo.
Esta tarea llevó un poco más de lo previsto, aproximadamente 2 meses y medio.

"Test" del prototipo

Se desarrolló otra implementación del mismo algoritmo implementado en la fase anterior, pero utilizando otra solución.
En esta fase se realizó el "test" de los dos algoritmos implementados y de la plataforma. Sirvió además para descubrir aspectos que eran importantes en la plataforma. Por ejemplo, la necesidad de ejecutar los algoritmos en diferentes hebras, permitir editar los calculation dags en la aplicación y ejecutar los algoritmos faltantes al crear la tabla comparativa de resultados.
La duración de esta fase fue aproximadamente la estimada.

Análisis y Diseño detallados

El objetivo de esta fase era el de rediseñar los componentes de la aplicación para permitir agregar nuevos algoritmos a la plataforma.
A partir de lo analizado en la fase anterior, se realizaron modificaciones en el diseño para brindar nuevas funcionalidades a la plataforma, planificándose la implementación de las mismas.
Durante esta fase se implementaron las funcionalidades que se querían agregar y se realizaron las modificaciones para permitir agregar nuevos algoritmos.
La duración de esta fase fue de aproximadamente 3 meses.

Implementación

Se implementó un nuevo algoritmo. El algoritmo consistía en una generalización de los dos algoritmos antes implementados. El algoritmo fue agregado con éxito a la plataforma.
Se implementaron las nuevas funcionalidades de la plataforma que fueron diseñadas en la fase anterior. Se le pudo dedicar más tiempo debido a que no se implementó otro algoritmo diferente a los ya implementados, como estaba planteado. Esto trajo consigo que se implementaran funcionalidades de la plataforma que no estaban previstas en el alcance inicial.
Se trabajó también en la primera versión del manual de usuario de la aplicación
La duración de la fase fue de aproximadamente 1 mes y medio.

Finalización

Se llevó a cabo de acuerdo a lo planificado. Se terminó con la documentación de la aplicación (diseño y manual de usuario) y se escribió el informe final del proyecto (este documento).

4.3 Gantt Comparativo

La figura 4.3.1 muestra una comparación entre la planificación original y la finalmente ejecutada.

Figura 4.3.1 – Gantt comparativo

5 CONCLUSIONES

En este documento se presentó el diseño y la implementación de una plataforma para evaluar la calidad de los datos devueltos por diferentes alternativas de cálculo. En esta sección se presentan los resultados obtenidos, los aportes y los trabajos futuros.

5.1 Resultados obtenidos

Al finalizar el proyecto se cuenta con una plataforma estable que permite la representación de diversas alternativas de cálculo y la ejecución de tres algoritmos realizados como casos de estudio. Este era el objetivo principal del proyecto y consideramos que se ha cumplido. Los puntos que revisten mayor importancia acerca de esta plataforma son:

- El diseño realizado sobre la representación de las alternativas de cálculo no fue modificado en el transcurso del proyecto. Este diseño permitió representar correctamente los casos de estudio tratados.
- La plataforma permite la visualización gráfica de las alternativa de cálculo con sus propiedades.
- Se cuenta con una forma sencilla de agregar nuevos algoritmos, inclusive en tiempo de ejecución.
- Quedó pendiente la implementación de un algoritmo radicalmente diferente de forma de validar la generalidad de la plataforma.
- La tabla que se presenta para comparar los resultados de los algoritmos presenta la información de una forma básica, no permitiendo un análisis detallado de los resultados. Para ello se permite guardar el resultado en una base de datos para su posterior estudio con herramientas diseñadas con ese fin.

5.2 Aportes

El principal aporte de este proyecto consiste en la plataforma obtenida, la cual es de utilidad para realizar de una forma amigable el "test" de los nuevos algoritmos que se desarrollen en el marco del trabajo de investigación llevado a cabo por el grupo CSI.

5.3 Extensiones y Trabajos futuros

Para realizar un "test" exhaustivo de la plataforma y verificar que se adapta a los diferentes casos de estudio sería interesante la implementación de un algoritmo sustancialmente diferente a los ya implementados.

Esta plataforma puede ser utilizada en diferentes ámbitos para evaluar propiedades de calidad. A su vez, los algoritmos que se implementan pueden ser compartidos entre los diferentes usuarios. Seria interesante brindar una interfaz web de la aplicación, manteniendo la lógica ya implementada, para facilitar a los usuarios el testeo e intercambio de algoritmos. A su vez se podrían brindar servicios web para la ejecución de los algoritmos.

Una actividad, que no se planteaba como objetivo de este proyecto, es la de analizar los resultados obtenidos.

El problema general de análisis es mucho más amplio y abarca otras áreas, como la obtención de las diferentes alternativas de cálculo o algoritmos para procesar los resultados obtenidos, que no son tratados en este proyecto. Aquí se plantea únicamente la construcción de una plataforma que permite ejecutar los algoritmos y mostrar sus resultados. Para la comparación de los resultados se presenta una tabla básica y se guardan los resultados en una base de datos para analizarlos con herramientas diseñadas para ese fin. Se podrían incorporar en la plataforma algoritmos que realicen la tarea de análisis de los resultados, así como también mejores informes de los resultados.

REFERENCIAS

[Per03] Peralta, V.: "*Freshness in a Materialization Context*". Working notes. Laboratoire PRiSM, Université de Versailles. Versailles, France. February 2003.

[Per03a] Peralta, V.: "*Calculating Optimal Refresh Frequencies to assure Data Freshness*". Working notes. Laboratoire PRiSM, Université de Versailles. Versailles, France. March 2003.

[Per03b] Peralta, V.: *"A general representation of mediation system"*. Working notes. Laboratoire PRiSM, Université de Versailles. Versailles, France. September 2003.

[Per04] Peralta, V. : *"Evaluation and enforcing data freshness in a Multi-Source information system*". *"*. Working notes. Laboratoire PRiSM, Université de Versailles. Versailles, France. March 2004.

[Ked99] Kedad, Z. Bouzeghoub, M.: "Discovering View Expressions from a Multi-Source Information System". Proc. 4th. Int. Conf. on Cooperative Information Systems (CoopIS), Edinburgh, Scotland, 1999.

[Nau98] Naumann, F.: "*Data fusion and data quality*". In Proc. of the New Techniques and Technologies for Statistics Seminar (NTTS'98). Sorrento, Italy. 1998.

"Software Engineering", Shari Pfleeger

"Applying UML and Patterns", Craig Larman

"Core java 2" (Volumen 1 y 2), Cay S. Horstmann – Gary Cornell.