Un Framework Extensible para la Integración de Razonamiento Basado en Casos con Aplicaciones Empresariales Orientadas a Objetos

María Celeste Carignano[1], Silvio Gonnet[2], Horacio Leone[2]

[1]Facultad Regional Santa Fe
Universidad Tecnológica Nacional
Lavaise 610, Santa Fe, Argentina
mcarigna@frsf.utn.edu.ar

[2]INGAR, Instituto de Desarrollo y Diseño
Universidad Tecnológica Nacional, CONICET
Avellaneda 3657, Santa Fe, Argentina
{sgonnet, hleone}@santafe-conicet.gov.ar

Resumen

En la vida cotidiana, las personas utilizan su experiencia para resolver las situaciones que se presentan, recordando circunstancias similares y adaptando las soluciones en función de las características de cada problema particular. En el contexto empresarial, estas prácticas también se llevan a cabo. Muchas veces el conocimiento asociado a ciertas experiencias se encuentra distribuido en las aplicaciones existentes, por lo que solo es necesario identificarlo y definir mecanismos apropiados para poder recuperarlo y reutilizarlo.

Razonamiento Basado en Casos es un paradigma de resolución de problemas que involucra el uso de experiencias pasadas para comprender y resolver nuevas situaciones. Se sustenta en la idea de que si una experiencia pasada fue exitosa puede ser utilizada en el presente, en su totalidad o como punto de partida, para decidir o resolver una nueva situación similar. Su principal hipótesis es que problemas similares tienen soluciones similares.

En este trabajo, se presenta un framework cuyo objetivo es facilitar la implementación de Razonamiento Basado en Casos en aplicaciones empresariales orientadas a objetos con el fin de poder hacer uso de la experiencia y conocimiento adquirido en el pasado para solucionar y resolver situaciones actuales con calidad y consistencia.

1. Introducción

Razonamiento Basado en Casos (RBC) es un paradigma de resolución de problemas [1] que involucra el uso de experiencias pasadas para comprender y resolver nuevas situaciones [2]. La principal hipótesis de RBC es que problemas similares tienen soluciones similares. La información y el conocimiento de situaciones similares previas sirven para resolver nuevos problemas.

RBC se sustenta en la idea de que si una experiencia pasada fue exitosa puede ser utilizada en el presente, en su totalidad o como punto de partida, para decidir o resolver una nueva situación similar.

Los beneficios potenciales de RBC son [3]:

- Descubrir conocimiento en los datos,
- Distribuir decisiones consistentes a lo largo de la organización,
- Preservar el "know-how" de los especialistas capturando sus experiencias,
- Transferir experiencias desde especialistas a novatos,
- Construir una memoria corporativa compartiendo experiencias individuales.

El uso de la memoria en la resolución de problemas no es un descubrimiento de RBC. RBC intenta reflejar el uso humano de los problemas y soluciones recordadas como punto de partida para la resolución de nuevas situaciones [4].

A diario las personas utilizan su experiencia para resolver las situaciones que se presentan, recordando circunstancias similares y adaptando las soluciones en función de las características de cada problema particular.

En el contexto empresarial, diversas decisiones pueden ser tomadas basándose en conocimiento e información de experiencias pasadas. Dicha información puede estar en la memoria de los tomadores de decisiones, o distribuida en las aplicaciones que se emplean en la organización. En este último caso, es posible implementar RBC para dar soporte a las personas en sus tareas.

A lo largo del tiempo, RBC ha sido aplicado en diferentes áreas, como ser leyes, medicina, arquitectura, mesa de ayuda, sistemas de planeamiento, de scheduling, entre otros. Actualmente, existen implementaciones académicas e industriales de RBC para dominios específicos ([5], [6], [7], [8], entre otros), así como también frameworks que permiten crear aplicaciones completas de RBC ([9], [10], [11], [12], [13], entre otros). Sin embargo, estas aplicaciones existen de forma aislada con respecto a los sistemas propios de una organización, ignorando las bases de conocimiento que se nutren a diario con la operatoria regular de las empresas.

En este trabajo, se presenta un framework cuyo objetivo es facilitar la implementación de RBC en aplicaciones empresariales orientadas a objetos existentes, con el fin de poder hacer uso de la experiencia y conocimiento adquirido en el pasado para solucionar y resolver situaciones actuales con calidad y consistencia. En el framework propuesto se presentan algunas simplificaciones, planteando como trabajos futuros la inclusión de aspectos más complejos de RBC.

En la siguiente sección, con el objetivo de facilitar la comprensión de los motivos de la propuesta, se presenta un ejemplo real en el cual podría implementarse la integración presentada en este trabajo. A continuación, se describen brevemente las principales características de RBC, se detalla el framework presentado y la implementación del framework en el contexto del ejemplo descrito previamente. Luego, se analiza bibliografía relacionada para finalizar exponiendo las conclusiones y aspectos sobre los que se continuará trabajando.

2. Introducción al ejemplo de aplicación. Descripción del contexto.

El objetivo de esta sección es describir el contexto en el cual se podría aplicar la propuesta presentada en este trabajo. Se presenta un ejemplo basado en un sistema real que pertenece a una empresa dedicada a la venta de neumáticos y reparación de automóviles. Dicho sistema está encargado de gestionar órdenes de servicio en el taller de reparación. Cuando un vehículo llega al taller se genera una orden de servicio, registrando los servicios que los mecánicos deben efectuar al vehículo en cuestión. A medida que se van realizando los servicios, los mecánicos dejan constancia en el sistema de que el servicio ha sido finalizado e indican si hubo algún inconveniente o aspecto en particular que deba ser informado al cliente. Finalmente, una constancia de servicio es generada e impresa para ser entregada al cliente como prueba de los servicios efectuados en su automóvil. Además, el sistema permite generar reportes con información sobre los servicios más requeridos, la cantidad de servicios realizados por cada empleado, cantidad de servicios realizados por día, etc.

Debido a que los mecánicos registran cuándo comienzan a realizar un servicio y cuándo terminan, es posible conocer el tiempo exacto que dedicaron a la realización de cada servicio en particular.

Teniendo en cuenta esta información, se detectó la oportunidad de poder emplearla para estimar cuánto tiempo tendría que esperar un cliente para que su vehículo esté listo.

Los servicios se clasifican en dos grupos de servicios: alineación y balanceo. Un cliente puede solicitar servicios de alineación (de uno o más neumáticos), servicio de balanceo, o ambos. La empresa cuenta con un mecánico especializado en cada grupo de servicios definido, los cuales serán identificados a partir de ahora como ***MecB*** y ***MecA***, según se trate del mecánico especializado en servicios de balanceo o de alineación, respectivamente. De esta forma, según los servicios que requiera el automóvil, su atención puede estar asignada a un mecánico en particular o a los dos. El orden en que se llevan a cabo los servicios es: primeros los servicios de balanceo, y finalmente el servicio de alineación.

Por lo tanto, cuando un cliente solicita servicios para su automóvil, el tiempo de espera estará conformado por:

- Si solo solicita servicios de balanceo: el tiempo en cola esperando que se desocupe ***MecB*** (T1) y el tiempo de realización de los servicios de balanceo en su vehículo (T2).

  Tiempo de espera = T1 + T2

- Si solo solicita el servicio de alineación: el tiempo en cola esperando que se desocupe ***MecA*** (T1), el tiempo de realización del servicio de alineación de su vehículo (T2).

  Tiempo de espera = T1 + T2

- Si solicita ambos servicios: el tiempo en cola esperando que se desocupe ***MecB*** (T1), el tiempo de realización de los servicios de balanceo en su vehículo (T2), el tiempo en cola esperando que se

desocupe ***MecA*** (T3), el tiempo de realización del servicio de alineación de su vehículo (T4).

Tiempo de espera = T1 + T2 + T3 + T4

Por lo tanto, para poder realizar el cálculo del tiempo de espera es necesario conocer cuánto tiempo estará ocupado cada mecánico con cada vehículo que debe atender. No existe una fórmula que sea fácilmente codificable para poder calcular el tiempo que le lleva a un mecánico satisfacer una orden de servicio, ya que depende de:

- el empleado: su experiencia, capacidades y habilidades,
- características propias del vehículo, como ser su tipo: automóvil o utilitario, y
- los servicios contratados, entre otras cosas.

Medir la capacidad que tiene un empleado para dar respuesta a la solicitud de un servicio es muy difícil. Por lo tanto, se considera de utilidad integrar la aplicación con el framework propuesto en este trabajo. De esta manera, se estaría emulando de manera automática la siguiente conversación:

- **Pedro** (empleado de atención al público): "¿José cuánto tiempo te llevaría hacer el servicio de alineación en el automóvil Peugot 406 de Juan Perez?"
- **José** (Mecánico): "las alineaciones generalmente las hago en una hora, pero con el auto de Juan Perez demoro treinta minutos más ya que también tiene reguladores para alinear en el eje trasero.".
- **Pedro** (empleado de atención al público): "Bueno, hay dos automóviles esperando, así que estaría listo en tres horas y media aproximadamente."

3. El ciclo de RBC

En RBC un caso denota una situación experimentada previamente, que ha sido capturada y aprendida de forma tal que puede ser reutilizada en la resolución de problemas futuros [1]. Suele estar compuesto por [14]:

- el problema: que describe el estado del mundo cuando ocurre el caso,
- la solución: que establece la solución encontrada.

RBC involucra la ejecución de cuatro actividades conocidas como las *4Res*: *Recuperar*, *Reusar*, *Revisar* y *Retener*, las cuales son mostradas en la Fig. 1.

La primera actividad del ciclo es la de **RECUPERACIÓN** de los casos más similares. Comienza con la descripción de un problema (conocido como nuevo caso) y finaliza con la obtención de uno o más casos recuperados. Cada caso recuperado propone soluciones previamente aplicadas para resolver un problema similar. Esta actividad generalmente es la más compleja y la que más estudio previo requiere dentro del ciclo de RBC. Involucra la evaluación de todos los casos en memoria y la determinación del grado de similitud entre cada uno de ellos y el nuevo caso.

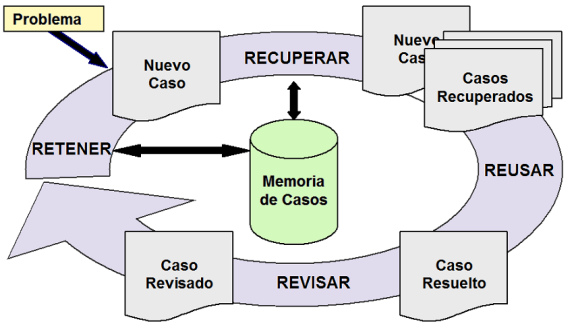

Figura 1. Actividades del proceso de Razonamiento Basado en Casos. Figura adaptada de [1].

Generalmente es llevada a cabo en dos pasos [2]:

- primero se recupera un conjunto de casos candidatos. Los casos candidatos son casos con potencial para que a partir de ellos se efectúen predicciones relevantes sobre el nuevo caso, y
- luego, se realiza un procesamiento para seleccionar el o los mejores casos del conjunto de casos candidatos previamente recuperados.

El algoritmo de recuperación utilizado para llevar a cabo esta actividad depende de cada aplicación de la técnica en particular, de la organización de la memoria de casos y del contenido y estructura de los mismos.

La segunda actividad tiene como objetivo **REUSAR** la información y el conocimiento del caso recuperado para resolver el nuevo problema. Es la responsable de proporcionar una solución (conformando un caso resuelto) para un nuevo problema a partir de las soluciones del caso recuperado [4]: soluciones antiguas son utilizadas como inspiración para resolver nuevos problemas [2]. Esta actividad puede ser muy sencilla y requerir solamente el retorno de las soluciones recuperadas (reutilización por copia) o lo suficientemente compleja como para necesitar que se realicen adaptaciones de las soluciones recuperadas para que sean aplicables al nuevo caso. En general, las adaptaciones se pueden clasificar en [14]:

- adaptaciones estructurales: las reglas de adaptación son aplicadas directamente desde la solución almacenada en el caso [15], y
- adaptaciones derivacionales: se reusan los algoritmos, métodos o reglas que generaron la

solución original para producir o derivar una solución para el nuevo problema.

La tercera actividad es la de **REVISIÓN** de la solución propuesta. Una vez que el nuevo caso ha sido resuelto debe ser probado en el mundo real. Cuando la solución generada en la actividad de reuso no es correcta surge una oportunidad para aprender de las fallas. De esta forma, durante la actividad de revisión, se llevan a cabo dos tareas [1]:

- la evaluación del caso resuelto generado por la actividad de reuso, y
- la reparación del caso resuelto, si es necesario, utilizando conocimiento específico del dominio.

Durante última actividad, la de **RETENCIÓN**, el caso revisado es incorporado a la memoria de casos como un caso aprendido con el objetivo de que esté disponible si un nuevo problema arriba para comenzar con el ciclo nuevamente.

Es importante aclarar que las actividades de reuso, revisión y retención raras veces ocurren sin la intervención humana [14].

4. Descripción del framework propuesto

El objetivo del framework propuesto, llamado ***ExCBR*** (**Ex**tensible **C**ase **B**ase **R**easoning), es dar soporte a las actividades de RBC, integrándose a las aplicaciones existentes en una organización que han sido desarrolladas con tecnología orientada a objetos.

Un aspecto clave de ***ExCBR*** es su capacidad para brindar soporte a la integración sin requerir la duplicación de la información de la organización ni la modificación de su estructura, ya que trabaja con los datos de las aplicaciones con las que se integra.

Para efectuar la integración se proponen un conjunto de clases e interfaces que permiten modelar la estructura y el comportamiento de un razonador de RBC. Debe tenerse en cuenta que en la versión del framework que se describe en este trabajo solo pueden ser representados casos para los cuales las actividades de recuperación, reuso, revisión y retención son automáticas, es decir, sin intervención humana.

Uno de los conceptos fundamentales del paradigma orientado a objetos sobre los que se construye este framework es el de *extensión*. La mayoría de las clases que pertenecen al framework deben ser extendidas para incluir el comportamiento necesario para llevar a cabo las actividades del ciclo RBC.

El primer paso para logar la integración de una aplicación orientada a objetos con el framework ***ExCBR*** es establecer cómo serán representados los casos e identificar cuál es el objeto que permite describir el caso en el contexto del dominio de la aplicación empresarial.

En ***ExCBR***, un caso puede ser representado de dos maneras:

a) De forma implícita: cuando la información de los casos se encuentra distribuida en la aplicación empresarial a la cual se integra la propuesta.
No hay una identificación explícita y estructural del problema y la solución que conforman el caso, sino que dichas partes están especificadas implícitamente en el comportamiento del razonador.
De ser así, se debe identificar el objeto de dominio que contiene la información del caso e implementar la interface: I*Case* en él (ver Fig. 2), para que pueda ser reconocido y manipulado durante el ciclo de RBC.

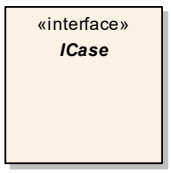

Figura 2. Representación implícita de casos en *ExCBR*

b) De forma explícita: mediante la identificación y construcción de las partes que lo constituyen: problema y solución. En la Fig. 3 se describe la estructura de un caso definido explícitamente.

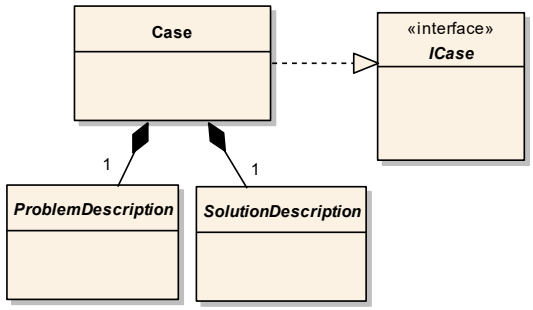

Figura 3. Representación explícita de casos en ExCBR

En este contexto, para describir un caso es necesario:

- Por un lado, definir clases particulares que extiendan las partes del caso: *ProblemDescription* y *SolutionDescription*,

respectivamente (ver Fig. 3), describiendo los datos que las conforman.

- Por otro lado, instanciar las clases *Case* y las extensiones de *ProblemDescription* y *SolutionDescription*, para que puedan ser empleadas durante el procesamiento de una consulta al razonador.

Cabe destacar que las instancias de *Case*, y de las extensiones de *ProblemDescription* y *SolutionDescription* no se corresponden con datos permanentes, es decir, corresponden a objetos temporales empleados durante un ciclo del razonador.

La estructura del razonador de ***ExCBR*** se presenta en la Fig. 4.

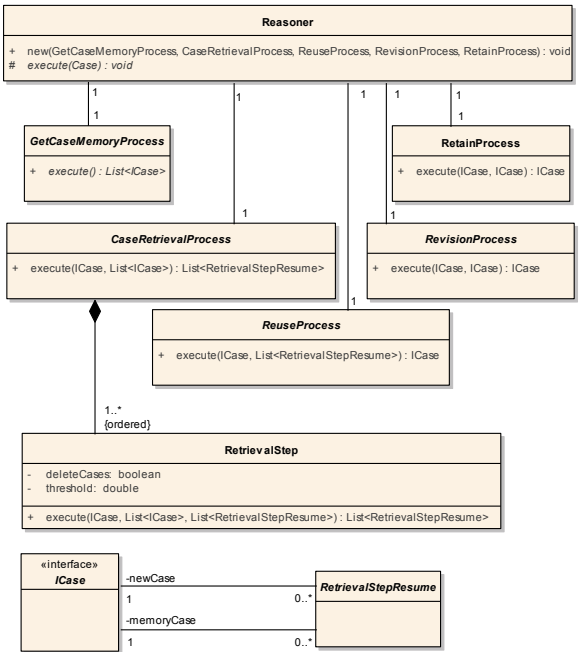

Figura 4. Clases involucradas en el razonador de *ExCBR*

Como puede observarse en la Fig. 4, existe una clase, llamada *Reasoner*, que tiene la responsabilidad de controlar la ejecución del ciclo de RBC. Para cada una de las actividades del ciclo: recuperación, reuso, revisión y retención, se encuentra definida una clase encargada del procesamiento correspondiente: *CaseRetrievalProcess*, *ReuseProcess*, *RevisionProcess*, *RetainProcess*. Además, se incluye una clase encargada de obtener o generar todos los casos de la memoria de casos: *GetCaseMemoryProcess*.

Cada una de estas clases debe ser extendida, para implementar el método *execute*. En las siguientes subsecciones se describen en detalle.

Al momento de crear el razonador, deben indicarse las instancias particulares de cada uno de los procesadores, esto se realiza mediante el constructor de *Reasoner*. En la Fig. 5 se presenta un ejemplo de implementación en lenguaje JAVA.

```
public Reasoner (GetCaseMemoryProcess
getCaseMemoryProcess, CaseRetrievalProcess
caseRetrievalProcess, ReuseProcess reuseProcess,
RevisionProcess revisionProcess, RetainProcess
retainProcess) {
   this.getCaseMemoryProcess =
                  getCaseMemoryProcess;
   this.caseRetrievalProcess =
                  caseRetrievalProcess;
   this.reuseProcess = reuseProcess;
   this.revisionProcess = revisionProcess;
   this.retainProcess = retainProcess;
}
```

Figura 5. Ejemplo de constructor de *Reasoner*.

Ante una consulta, el razonador invocará a la instancia responsable de cada actividad para poder resolver el problema actual. En la Fig. 6 se presenta un ejemplo de implementación en lenguaje JAVA, para facilitar la comprensión de su comportamiento.

```
public ICase execute(ICase newCase) {
   List<ICase> caseMemory =
                  getCaseMemoryProcess.execute();

   List<RetrievalStepResume> retrievalResume =
                  caseRetrievalProcess.execute(
                  newCase, caseMemory);

   ICase solvedCase = reuseProcess.execute(
                  newCase, retrievalResume);

   ICase revisedCase = revisionProcess.execute(
                  newCase, solvedCase);

   ICase learnedCase = retainProcess.execute(
                  newCase, revisedCase);

   return learnedCase;
}
```

Figura 6. Ejemplo de implementación del razonador.

4.1. Proceso: obtención de casos de la memoria.

Las instancias de las clases que extienden a *GetCaseMemoryProcess* tienen la responsabilidad de obtener todos los casos de la memoria de casos. La obtención de los casos puede realizarse de varias formas:

- Recuperando todos los objetos que implementan la interfaz *ICase,* o

- Creando los objetos *Case* en función de las características particulares del problema a resolver.

La lógica asociada a esta recuperación debe ser explicitada en el método *execute*. En dicho procedimiento se deberán contemplar cuestiones asociadas a performance y consistencia de los datos manipulados, teniendo en cuenta que la base de casos, o la fuente de información, puede ser de un gran volumen.

4.2. Proceso: recuperación de casos.

Las instancias de las clases que extienden a *CaseRetrievalProcess* están encargadas de recuperar de la memoria de casos el o los casos que presentan mejor similitud con el nuevo caso. La lógica asociada a la recuperación se debe describir en el método *execute*.

Para llevar a cabo esta actividad, se propone la definición del procesamiento mediante una secuencia ordenada de pasos (extensiones de *RetrievalStep* en Fig. 4), los cuales tiene acceso a:

- El nuevo caso,
- Los casos de la memoria
- El resultado del procesamiento del caso anterior, el cual se almacena temporalmente en objetos de tipo *RetrievalStepResume* (ver Fig. 4).

Se propone la utilización de estos objetos temporales para facilitar el procesamiento de actividades de recuperación complejas, que constan de varias etapas, por ejemplo:

- Un emparejamiento inicial para recuperar los casos candidatos, y
- Una selección posterior para encontrar el caso que mejor se ajusta.

En aquellas implementaciones de RBC en donde la actividad de recuperación no reviste complejidad, el procesamiento puede ser realizado en un único paso.

En estos pasos se debe plasmar el algoritmo de recuperación junto con la medida de similitud seleccionada para cada problema en particular.

En la Fig. 7 se presenta un ejemplo de implementación en lenguaje JAVA, para facilitar la comprensión de su comportamiento.

4.3. Proceso: reuso, revisión y retención.

Como se mencionó anteriormente, esta versión del framework solo contempla reuso, revisión y retención automáticos, es decir, sin intervención humana.

La lógica asociada al reuso, la revisión y la retención se debe describir en el método *execute* de la clase correspondiente (*ReuseProcess*, *RevisionProcess* y *RetainProcess*, respectivamente).

```
public class OneCaseRetrievalProcess {

   private List<RetrievalStep> retrievalSteps;

   public List<RetrievalStepResume> execute(
        ICase newCase, List<ICase> caseMemory) {

      List<RetrievalStepResume> previousStepResume=
              new ArrayList<RetrievalStepResume>();

       for (RetrievalStep step: retrievalSteps) {

            previousStepResume = step.execute(
                                 newCase, caseMemory,
                                 previousStepResume);
        }

        return previousStepResume;
    }

}
```

Figura 7. Ejemplo de implementación de la recuperación de casos.

4.3.1. Proceso: reuso

En el caso del reuso, éste puede involucrar la copia de la solución del caso recuperado en el nuevo caso, o la adaptación de la solución del caso recuperado para ser asignado en el nuevo caso, o un procesamiento más complejo si se recuperan más de un caso en la actividad de recuperación.

La decisión sobre qué tipo de reuso implementar, depende de la situación particular a la que se intenta dar solución.

4.3.2. Proceso: revisión

Con respecto a la revisión, se deben llevar a cabo tareas de evaluación de la solución propuesta en la actividad de reuso, y en caso de que se detecte alguna falla o problema, se puede incluir la implementación de la solución. En esta versión del framework solo se contempla la revisión automática, por lo que no se podrán considerar evaluaciones que dependan de actores humanos, sino que únicamente se podrán llevar a cabo aquellas que puedan ser resueltas en tiempo de ejecución mediante procedimientos previamente codificados.

4.3.3. Proceso: retención

Finalmente, la actividad de retención incorpora la nueva solución al nuevo caso que fue recibido por parámetro. Esta actividad no necesariamente impactará en los datos persistentes de la aplicación, pero si devolverá el caso actualizado para que el objeto que

invocó el razonador con la consulta pueda tomar decisiones en base a esa información.

5. Integración en el marco del ejemplo de aplicación

En esta sección se describirá una posible implementación del framework ***ExCBR*** en el sistema de gestión de órdenes de servicio presentado en la sección 2. En dicho sistema, una orden de servicio se representa mediante las clases detalladas en la Fig. 8.

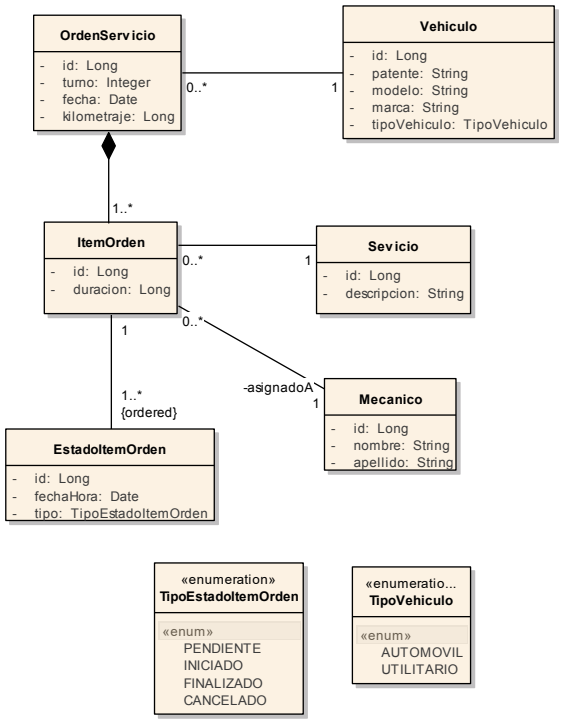

Figura 8. Descripción de una orden de servicio.

Una orden (*OrdenServicio* en Fig. 8) está compuesta por ítems (*ItemOrden* en Fig. 8) que describen los servicios (*Servicio* en Fig. 8) que deben llevar a cabo los mecánicos (*Mecanico* en Fig. 8) para un vehículo determinado (*Vehiculo* en Fig. 8). El tiempo empleado por cada mecánico para realizar un servicio puede calcularse considerando los cambios de estado de un ítem de la orden (*EstadoItemOrden* en Fig. 8):

- cuando el ítem cambia de estado *Pendiente* a estado *Iniciado* se considera que el mecánico comienza a trabajar en el servicio, y
- cuando cambia de estado *Iniciado* a estado *Finalizado*, se considera que el mecánico termina de trabajar en el servicio.

Por lo tanto, conociendo la hora en la que se efectúa cada cambio de estado es posible calcular el tiempo dedicado por el mecánico para realizar el servicio. Para facilitar la presentación de este trabajo se considera que el atributo *duración* de *ItemOrden* contiene dicha información.

Al emplear el framework ***ExCBR*** para integrar el sistema con RBC se debe decidir cuál será la estructura que tendrán los casos manipulados. Del análisis presentado, se puede observar que el concepto de caso se corresponde con un ítem de una orden de servicios. Por lo que las opciones posibles de realización son:

- Opción 1 (de forma explícita): identificando todas las partes del caso, como se muestran en la Fig. 9.

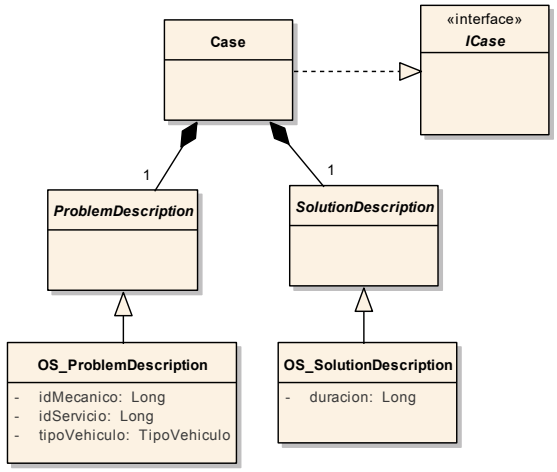

Figura 9. Caso representando una orden de servicio.

- Opción 2 (de forma implícita): que *ItemOrden* implemente la interfaz *ICase*, como se muestra en la Fig. 10.

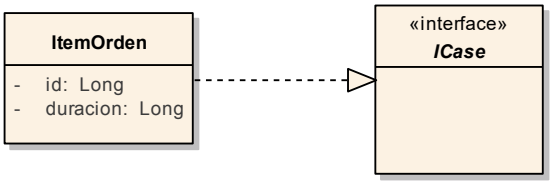

Figura 10. Orden de servicio como caso.

En las siguientes subsecciones, se describen las características particulares de la integración considerando cada una de las representaciones de casos enumeradas previamente.

De esta forma, ante el arribo de un nuevo cliente, se deberán efectuar varias consultas al razonador según los servicios que el cliente solicite para su automóvil

(identificado como ***A***). Por ejemplo, si el cliente solicita servicios de alineación y balanceo y hay un automóvil en cola (identificado como ***X***) para recibir servicios de balanceo y dos automóviles en cola (identificados como ***Y*** y ***Z***) para recibir servicios de alineación, para determinar el tiempo de espera del nuevo vehículo, se deberá consultar al razonador para conocer la siguiente información:

- Cuánto tiempo estarán ocupados los mecánicos con otros automóviles para poder comenzar a trabajar con ***A***:
  - Cuánto tiempo necesita el mecánico ***MecB*** para realizar el servicio de balanceo en ***X***,
  - Cuánto tiempo necesita el mecánico ***MecA*** para realizar el servicio de alineación en ***Y***,
  - Cuánto tiempo necesita el mecánico ***MecA*** para realizar el servicio de alineación en ***Z***,
  - Cuánto tiempo necesita el mecánico ***MecA*** para realizar el servicio de alineación en ***X***.
- Cuánto tiempo estarán ocupados los mecánicos ***MecB*** y ***MecA*** trabajando con ***A***:
  - Cuánto tiempo necesita el mecánico ***MecB*** para realizar el servicio de balanceo en ***A***,
  - Cuánto tiempo necesita el mecánico ***MecA*** para realizar el servicio de alineación en ***A***.

Con toda esta información, y teniendo en cuenta que los mecánicos ***MecB*** y ***MecA*** trabajan en paralelo, se puede conocer el tiempo de espera del cliente propietario del vehículo ***A***.

5.1. Integración considerando la representación explícita de casos

Si se selecciona la Opción 1 para representar los casos en el contexto del ejemplo de aplicación, el proceso *GetCaseMemoryProcess* deberá construir los objetos *Case* de la Fig. 10 para cada ítem de cada orden de servicio del sistema. De esta forma, los casos que representan una consulta particular tendrían la forma presentada en la Fig. 11.

El proceso de *recuperación*, implementado en la clase que extiende a *CaseRetrievalProcess,* se divide en dos pasos:

- Primer paso: se seleccionan todos los casos de la memoria que involucran al mecánico y servicio del nuevo caso. Se obtiene como resultado una lista de casos, identificada como *ListaPasoUno*.
- Segundo paso: se analiza si entre los casos recuperados en el primer paso (de *ListaPasoUno)*, existen casos para el mismo tipo de vehículo que el nuevo caso. De ser así, se conforma una lista con esos casos, llamada *ListaPasoDos.* Caso contrario, la *ListaPasoDos* contendrá los mismos casos recibidos en la *ListaPasoUno*. Como resultado de esta actividad se retornan dos casos: el de mayor duración y el de menor duración.

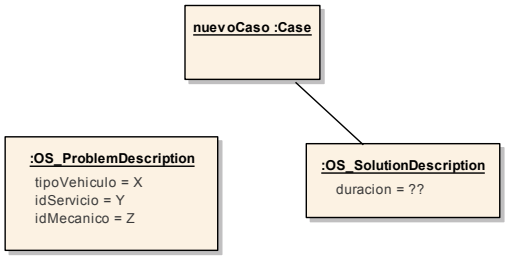

Figura 11. Ejemplo de objetos de un nuevo caso.

Como resultado de este proceso, se obtiene una lista conformada por cero, uno o dos objetos de tipo *RetrievalStepResume*, generados a partir del nuevo caso y los casos de *ListaPasoDos*. Si no existen casos para ese empleado y servicio, se retorna una lista vacía.

El proceso de *reuso*, implementado en alguna extensión de *ReuseProcess*, tiene como responsabilidad asignar una duración al nuevo caso, para conformar un caso resuelto. Un caso se considera resuelto cuando puede conocerse la duración del servicio, la cual se calcula como el promedio de las duraciones de los casos recuperados.

En el contexto de este sistema, y este problema, no se definen acciones dentro de la actividad de revisión ni retención, es decir que el método *execute* de las clases que extienden las clases *RevisionProcess y RetainProcess* solo deben copiar en la salida los casos recibidos. Esto se debe a que:

- Dadas las características del problema, no se deben llevar a cabo tareas de revisión y,
- Dado que los casos estructurados no se almacenan de forma persistente, tampoco se necesita retener ningún valor, solo retornarlo al componente del sistema de órdenes de servicio que invocó al razonador.

5.2. Integración considerando la representación implícita de casos

Si se selecciona la Opción 2 de representación de casos en el contexto del ejemplo de aplicación, el proceso *GetCaseMemoryProcess* solo deberá recuperar todos los ítems de todas las ordenes de servicio del sistema. Dado que *ItemOrden* implementa la interface *ICase*, puede ser empleado en todo el ciclo de razonamiento. De esta forma, los casos que representan una consulta particular tendrían la forma presentada en la Fig. 12.

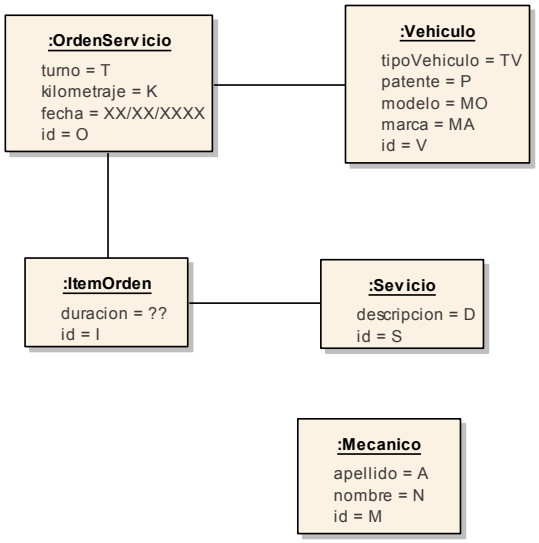

Figura 12. Ejemplo de representación implícita de casos.

El proceso de *recuperación*, implementado en la clase que extiende a *CaseRetrievalProcess,* se divide en dos pasos:

- Primer paso: se seleccionan todos los objetos *ItemOrden* que están asociados al mecánico y servicio que originan la consulta. Se obtiene como resultado una lista de objetos *ItemOrden*, llamada *ListaPasoUno*, que cumplen la condición de selección.
- Segundo paso: se analiza si entre las instancias de *ItemOrden* recuperadas en el primer paso, existen instancias de *ItemOrden* asociadas a órdenes creadas sobre vehículos del mismo tipo que el que origina la consulta. De ser así, se conforma una lista con esos objetos *ItemOrden*, llamada *ListaPasoDos.* Caso contrario, la *ListaPasoDos* contendrá los mismos objetos recibidos en la *ListaPasoUno*. Como resultado de esta actividad se retornan dos instancias de *ItemOrden*: la que tiene mayor duración y la que tiene menor duración.

Como resultado de este proceso, se obtiene una lista conformada por cero, uno o dos objetos de tipo *RetrievalStepResume*, generados a partir del objeto *ItemOrden* que origina la consulta y los objetos de *ListaPasoDos*. Si no existen *ItemOrden* asociados a ese empleado y servicio, se retorna una lista vacía.

Como puede observarse, con esta opción de representación la descripción de un caso se encuentra distribuida en la implementación del proceso de recuperación. La desventaja de esta opción es que genera una solución que requiere mayor esfuerzo en tiempo de mantenimiento, sin embargo, es útil en aquellas situaciones en las que definir una estructura de casos explícita implica replicar gran parte de las entidades del sistema que lo componen.

El proceso de *reuso*, implementado en alguna extensión de *ReuseProcess*, tiene como responsabilidad asignar una duración a la instancia de *ItemOrden* que originó la consulta. Dicha duración se calcula como el promedio de las duraciones de los objetos *ItemOrden* recuperados.

En el contexto de este sistema, y este problema, no se definen acciones dentro de la actividad de *revisión*, es decir que el método *execute* de la clase que extiende a la clase *RevisionProcess* solo debe copiar en la salida la instancia de *ItemOrden* recibida ya que no se deben llevar a cabo tareas de revisión.

Finalmente, el proceso de *retención* involucra la persistencia de la duración calculada del objeto *ItemOrden* que originó la consulta al razonador.

6. Discusión

Razonamiento Basado en Casos es una metodología que se encuentra dentro del área de Inteligencia Artificial, sobre la cual se han desarrollado numerosos trabajos y herramientas. Las áreas de aplicación de RBC van desde mesas de ayuda y servicio al cliente, sistemas de recomendación en comercio electrónico, gestión del conocimiento y experiencias, aplicaciones médicas, aplicaciones en procesamiento de imágenes, aplicaciones en leyes, diagnósticos técnicos, diseño, planeamiento, juegos de computadora, y música [16].

Actualmente, existen implementaciones académicas e industriales de RBC para dominios específicos ([5], [6], [7], [8], entre otros), así como también frameworks que permiten crear aplicaciones completas de RBC ([9], [10], [11], [12], [13], entre otros). En los trabajos de ElKafrawy y Mohamed [17] y Atanassov y Antonov [18], se presentan estudios que comparan distintas

herramientas de razonamiento basado en casos. En [17] se las divide en dos categorías: aquellas aplicaciones que tienen como objetivo generar implementaciones de RBC con interfaces de usuario gráficas, y aquellas aplicaciones que son implementaciones de RBC directas sobre un dominio específico para resolver un problema en particular.

Sin embargo, a diferencia de este trabajo, ninguna de las herramientas estudiadas plantea la integración con aplicaciones existentes.

Poder construir un framework a partir del cual se pueda integrar la metodología de RBC con aplicaciones empresariales permitiría ampliar el espectro de aplicaciones de RBC, ya que podría contribuir fácilmente con la reutilización del conocimiento y la experiencia documentada en las aplicaciones y sus bases de datos de forma implícita.

7. Conclusiones y trabajos futuros

En este trabajo se presentó un framework, llamado ***ExCBR***, que define un conjunto de clases e interfaces para facilitar la integración de la metodología de RBC con aplicaciones empresariales orientadas a objetos.

Uno de los objetivos específicos de este framework está relacionado con el hecho de evitar la duplicación de información o la necesidad de emplear nuevas aplicaciones para poder gozar de los beneficios de aplicar RBC en una organización. Como pudo observarse en el ejemplo desarrollado a lo largo del trabajo, fue posible integrar la metodología de RBC con una aplicación real, sin que fuera necesario extraer la información de la aplicación, convertirla a un formato que pudiera procesar una de las herramientas específicamente desarrolladas para RBC, procesar los datos para luego integrar los resultados con la aplicación o fuente de información de origen. La integración de la metodología con las aplicaciones permite gozar de los beneficios de RBC en tiempo de ejecución de las aplicaciones empresariales.

En este trabajo se presenta una versión inicial y reducida del framework, a partir de la cual se desprenden varias líneas de trabajo, como ser:

- La definición de las clases del framework en archivos xml, de forma tal que no sea necesario codificar todos los métodos *execute* de las clases que pertenecen al framework.
- La contemplación de las medidas de similitud más comunes en el proceso de recuperación.
- El manejo del ciclo de RBC de manera no sincrónica, de forma tal de eliminar la restricción actual de que las cuatro actividades del ciclo sean automáticas.

De la identificación de estas líneas de trabajo, se evidencia que la integración de RBC en aplicaciones existentes es una línea que aún no ha sido totalmente desarrollada y presenta un amplio campo de aplicación.

8. Agradecimientos

Este trabajo ha sido financiado en forma conjunta por la Universidad Tecnológica Nacional y CONICET. Se agradece el apoyo brindado por estas instituciones.

9. Referencias

[1] A. Aamodt, E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. AI Communications. IOS Press, 7(1):39–59, 1994.

[2] J. Kolodner. An introduction to case-based reasoning. *Artificial Intelligence Review*, 6:3–34, 1992.

[3] A., Klaus-Dieter. A review of industrial case-based reasoning tools. AI Intelligence, 1995.

[4] R. López de Mántaras, E. Plaza. Case-based reasoning: an overview. *AI Communications. IOS Press*, 10:21–29, 1997.

[5] A. Pla, B. López, P. Gay, C. Pous. eXiT*CBR.v2: Distributed case-based reasoning tool for medical prognosis. Decision Support Systems. Volume 54, Issue 3, February 2013, Pages 1499-1510.

[6] M. C. Carignano, S. Gonnet, H. Leone. RADS: Una herramienta para reutilizar estrategias en diseños de arquitecturas de software. Simposio Argentino de Ingeniería de Software, ASSE 2016 – 45 JAIIO Jornadas Argentinas de Informática, Páginas: 147-158, 2016. ISSN: 2451-7593.

[7] V. Ayzenshtadt, C. Langenhan, S.S. Bukhari, K.D. Althoff, F. Petzold, A. Dengel: Thinking with containers: a multi-agent retrieval approach for the case-based semantic search of architectural designs. In: Filipe, J., van den Herik, J. (eds.) 8th International Conference on Agents and Artificial Intelligence (ICAART-2016), pp. 24–26. SCITEPRESS, Rome, February 2016

[8] V. Ayzenshtadt, C. Langenhan, K.D. Althoff, S.S. Bukhari, F. Petzold, A. Dengel, (2017). Extending the Flexibility of Case-Based Design Support Tools: A Use Case in the Architectural Domain. 10.1007/978-3-319-61030-6_4.

[9] J. Recio-García, P. Gonzalez-Calero, B. Díaz-Agudo. jCOLIBRI2: A framework for building Case-based reasoning systems. Science of Computer Programming. 79. 126–145. 2014.

[10] K. Bach, K. Althoff, Klaus-Dieter. Developing Case-Based Reasoning Applications Using myCBR 3. 7466. 17-31. 2012.

[11] FreeCBR, documentado y disponible en http://freecbr.sourceforge.net/. Accedido en Octubre de 2017

[12] AIAI CBR Shell. Documentado y disponible en http://www.aiai.ed.ac.uk/project/cbr/CBRDistrib/. Accedido en Octubre de 2017.

[13] myCBR. Documentado y disponible en http://www.mycbr-project.net/index.html. Accedido en Octubre de 2017

[14] I. Watson, F. Marir. Case-based reasoning: A review. *The Knowledge Engineering Review*, 9(4):327–354, 1994.

[15] J. Kolodner. *Case-based Reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[16] K. Bach, M. Minor, M. Reichle, (2008). Case-based Reasoning Introduction and Recent Developments.

[17] P. ElKafrawy, R.A. Mohamed. Comparative Study Of Case Based Reasoning Software. International Journal of Scientific Research & Management Studies 1 (6), 224-233, 2014

[18] L. Atanassov, L. Antonov. A comparative analysis of case based reasoning software frameworks jColibri and myCBR. Journal of the University of Chemical Technology and Metallurgy, 47, 1, 2012, 83-90