DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo práctico Nro. 1

Programming in the small

6 de octubre de 2011 — Ingeniería del Software 2

Integrante	LU	Correo electrónico
Facundo Carrillo	693/07	facu.zeta@gmail.com
Rodrigo Castaño	602/07	castano.rodrigo@gmail.com
Brian Curcio	61/07	bcurcio@gmail.com
Federico Pousa	221/07	fedepousa@gmail.com
Felipe Schargorosdky	691/07	schargo88@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Introducción

La idea del presente trabajo práctico es realizar el diseño y la implementación correspondiente a un sprint de Scrum para un sistema de elecciones de Codep.

Para realizar esta tarea, se simulará un sprint de scrum, generando primero las users stories que corresponden al backlog de todo el producto, y luego seleccionando las users stories que pertenecen a las funcionalidades que se deben presentar en esta entrega.

Una vez generadas las users stories a tener en cuenta, se desarrolló el diseño total del sistema, abarcando todas las funcionalidades que se quieren tener en cuenta en este primer sprint.

Para realizar el diseño total del sistema, se fueron atacando cada funcionalidad por sí sola, y luego se fue combinando todo el diseño para tener una unificación final del sistema.

Luego, para reforzar el diseño de clases, se fueron desarrollando diferentes diagramas de clase y de objetos para entender mejor las relaciones tanto dinámicas como estáticas de los objetos relativos a las diferentes funcionalidades.

Por último, se implemento las funcionalidades deseadas en lenguaje Smalltalk.

2. Product Backlog

A continuación se detallan las Users Stories pertenecientes al presente proyecto. Las mismas se clasifican según las diferentes funcionalidades atacadas del sistema, división que luego resultará importante tanto para el diseño del sistema, como para la implementación del mismo.

Logueo al sistema

Como: Usuario
Quiero: Poder loguearme al sistema
De forma que: Pueda utilizar las diferentes opciones del mismo, como la votación.

Story points: 2

Bussines value: 50

Como: Usuario
Quiero: Que nadie se loguee con mi cuenta
De forma que: Nadie pueda candidatearme o votar por mi.

Story points: 3

Bussines value: 100

Como: Usuario
Quiero: Poder desloguearme de forma segura del sistema
De forma que: Nadie pueda utilizar mis datos o mi cuenta.

Story points: 1

Bussines value: 10

Como: Usuario
Quiero: Poder cambiar mi constraseña
De forma que: Pueda cambiar la constraseña que me dan por defecto.

Story points: 1

Bussines value: 10

Postularse como candidato

Como: Junta electoral
Quiero: Poder abrir etapa de postulaciones
De forma que: los usuarios interesados puedan candidatearse

Story points: 5

Bussines value: 20

Como: Aspirante a candidato
Quiero: Poder postularme como candidato
De forma que: Pueda presentarme en las siguientes elecciones

Story points: 2

Bussines value: 20

Como: Candidato
Quiero: Poder darme de baja
De forma que: Pueda arrenpetirme de postularme como candidato

Story points: 1

Bussines value: 10

Como: Junta electoral
Quiero: Que el proceso de postulación chequee los requerimientos del postulado
De forma que: Solo puedan postularse las personas que cumplan con los requisitos establecidos

Story points: 5

Bussines value: 50

Como: Junta electoral
Quiero: Poder cambiar el reglamento de postulación
De forma que: Los requerimientos puedan ser adaptables

Story points: 8

Bussines value: 75

Emisión de voto

Como: Junta electoral
Quiero: Poder abrir etapa de votación
De forma que: se termine la etapa de postulación y comience la votación de los candidatos

Story points: 5

Bussines value: 100

Como: Usuario
Quiero: Poder emitir mi voto
De forma que: mi voto quede registrado para la votación actual.

Story points: 1

Bussines value: 50

Como: Junta electoral
Quiero: Saber quien voto
De forma que: Pueda ir teniendo idea de que porcentaje de empadronados se presentaron

Story points: 1

Bussines value: 20

Como: Usuario
Quiero: Recibir un certificado de emisión de voto
De forma que: tenga una forma de demostrar mi participación en la votación

Story points: 2

Bussines value: 10

Como: Usuario
Quiero: Que mi voto sea anónimo
De forma que: Nadie se entere de mi voto

Story points: 2

Bussines value: 10

Finalización del comicio

Como: Junta electoral
Quiero: Poder cerrar el período de votación
De forma que: se de paso al escrutinio de los votos

Story points: 3

Bussines value: 20

Como: Junta electoral
Quiero: que se labre un acta con los resultados
De forma que: se obtenga una presentación formal de los resultados de la votación

Story points: 3

Bussines value: 100

Como: Junta Electoral
Quiero: Ser notificada de los conflictos en la designación de cargos
De forma que: se puedan resolver mediante mecanismos ad hoc

Story points: 2

Bussines value: 50

3. Sprint Backlog

En el sprint deberían ir las users stories que se van a atacar durante cierto período de tiempo. Este período de tiempo no se encuentra prefijado como en un ambiente de desarrollo, sino que depende de las fechas de entrega del trabajo práctico, sumado al hecho que la dedicación no es constante y periódica como lo sería en un ambiente de desarrollo.

Dado que así lo requiere el enunciado, que en este caso cumpliría el rol de un product owner, el presente sprint solamente contiene la users stories correspondientes a las funcionalidades que se quieren atacar en esta iteración. Por lo tanto, para este sprint se tuvieron en cuenta la users stories correspondientes a las funcionalidades de:

Logging

Como usuario quiero poder logear al sistema de manera que pueda utilizar las diferentes opciones del mismo, como la votación.

- Armar funcionalidad para logear
- Entrar al sistema logeado

Como usario quiero que nadie se logee a mi cuenta de manera que nadie pueda candidatearme a mi o votar por mi.

- Autenticación de constraseñas mediante un hash
- Mensaje de error en caso de contraseña incorrecta.

Postulación de candidatos

Como junta electoral quiero poder abrir la etapa de postulaciones de forma que los usuarios interesados puedan candidatearse

- Armar funcionalidad para iniciar etapa de postulaciones
- Verificar que el usuario corresponde a un miembro de la junta electoral

Como aspirante a candidato quiero poder posultarme como candidato de forma que pueda presentarme en las siguientes elecciones

- Armar funcionalidad para que el aspirante pueda candidatearse
- Registrar un nuevo candidato
- Testear el agregado de candidatos

Como junta electoral quiero que el proceso de postulación chequee los requerimientos del postulado de forma que solo puedan postularse las personas que cumplan con los requerimientos establecidos

- Validar el candidato utilizando las reglas para validar dicho candidato.
- Armar ventana de aviso si el candidato es inválido.

Como junta electoral quiero poder cambiar el reglamento de postulación de forma que los requerimientos puedan ser adaptables

- Crear validador de candidatos para cada claustro.
- Permitir que un usuario de junta electoral pueda cambiar el validador.

Votación

Como junta electoral quiero poder abrir la etapa de votación de forma que se termine la etapa de postualación y comienze la votación de candidatos

- Armar funcionalidad para cerrar postulaciones
- Armar funcionalidad para iniciar el inicio de los comicios
- Colectar los candidatos que fueron correctamente validados
- Verificar que el usuario es miembro de la junta electoral
- Testear el cierre de postulaciones

Como usuario quiero poder emitir mi voto de forma que mi voto quede registrado para la votación actual

- Verficar que el usuario no haya votado
- Verificar que el candidato no pueda votar a miembros no candidatos de su claustro
- Registrar el voto del usuario

Clausura del acto electoral y generación de resultados

Como junta electoral quiero poder cerrar el período de votación de forma que se de paso al escrutinio de votos

- Funcionalidad para cerrar los comicios
- Realizar el armado del escrutino por cada claustro

Como junta electoral quiero que se labre un acta con los resultados de forma que se obtenga una presentación formal de los resultados de la votación

- Decidir los candidatos que resutaron electos.
- Armar el acta con los resultados
- Testear funcionalidad del armado de acta

4. Product Increment

El primer paso para realizar la demo fue elegir el lenguaje que se utilizó. Decidimos utilizar Smalltalk debido a que habia miembros del equipo que estaban familiarizados con las herramientas para desarrollo y porque al ser un lenguaje de objetos resulta más natural modelar al mundo e implementar un sistema a partir de un diseño de objetos.

Las funcionalidades a implementar solicitadas por el enunciado son:

- Alta de candidatos
- Emisión de voto de un miembro del departamento
- Confección del acta final resultante de la elección

La demo consta de un conjunto de paquetes de Smalltalk que permite realizar las elecciones. La interacción la realizamos mediante el transcript. Inicialmente cargamos padrones de alumnos, profesores y graduados y creamos instancias de objetos que sirven como validadores de candidatos. Luego tenemos un objeto llamado elección que es el que será encargado de todo el proceso, es decir, para utilizar el sistema como un administrador lo haremos mediante este objeto.

Otro objeto con el que interactuaremos sera la interfaz de loggeador, que permitirá autenticar a los usuarios para realizar las distintas acciones que seran enviadas por medio de la interfaz correspondiente al período de elección en la que se encuentre la misma.

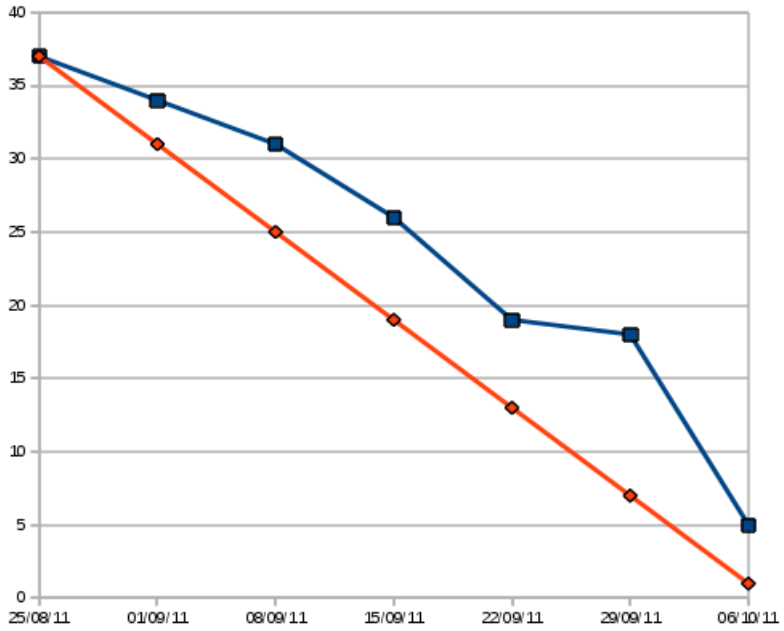
5. Burndown Charts

Sprint Burndown

En el sprint burndown mostramos como fuimos resolviendo las tareas en nuestro sprint basandonos en los story points como medida. Aca estamos considerando como terminada una tarea cuando ya teniamos la forma de resolver, aunque no necesariamente este implementado. Decidimos mostrarlo asi, ya que muestra mejor el proceso del desarrollo del trabajo práctico.

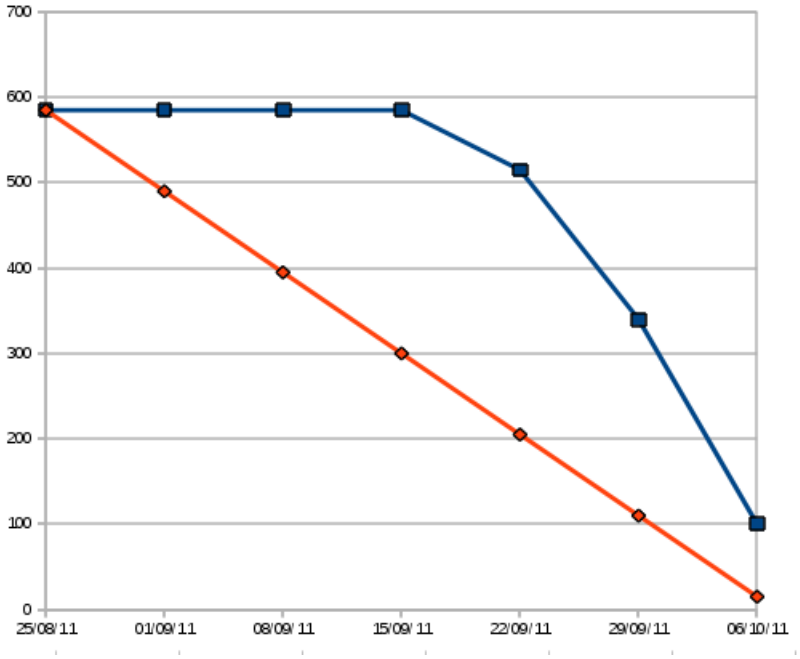
La linea rojo representa un proceso de desarrollo *ideal* que no representa necesariamente un proceso de desarrollo posible, pero sirve como guia para poder ver el desvio del desarrollo actual. En linea azul vemos el desarrollo del equipo durante el sprint. Cabe destacar, tal como mencionamos en la sección de sprint backlog, que nuestro sprint abarca el tiempo entre la presentación del tp y la fecha de entrega.

Se puede observar del gráfico que adquirimos más velocidad al acercarse la fecha de entrega. Esto en parte se debe tambien a que no mostramos tareas como la elaboración del sprint y product backlog. También vemos que durante la semana del parcial bajamos la velocidad debido a que no pudimos dedicarnos al trabajo práctico. Por último hubo una tarea del sprint backlog que al final decidimos que no era necesaria de implementar para mostrar la funcionalidad escencial.



Product Burndown

En el product backlog mostramos el avance de la implementación basandonos en el business value de cada story. Nuevamente mostramos en rojo un proceso de desarrollo *ideal* y en azul el proceso de desarrollo del equipo. Las primeras semanas tuvimos mucha inactividad debido a que estuvimos más tiempo pensando el diseño de objetos. Además nos falto implementar el story del hasheo de contraseñas porque consideramos que no era necesario para mostrar el funcionamiento para la demo.



6. Diseño Orientado a Objetos

A continuación se presenta el diseño orientado a objetos de todo el sistema implementado.

Dado que se quiere hacer hincapie en diferentes funcionalidades, se presentará primero el sistema en su totalidad y luego se irá explicando cada una de las partes más importantes.

Aca va el diagrama

Antes de pasar a la explicación de cada una de las funcionalidades, se fundamentaran algunas decisiones de diseño a un nivel más alto y no dentro de una funcionalidad en particular.

- Desacomplamiento de las funcionalidades: El sistema VOX es lo suficientemente chico como para poder implementarlo sin la necesidad de un buen diseño y aún así podría funcionar. Se podría hacer que existan pocos objetos que tengan muchas responsabilidades distintas sobre el sistema. Por ejemplo, podría haber un solo objeto que reifique a las elecciones en sí y que se encargue de la votación, de la postulación de candidatos y de todas las responsabilidades a implementar. Sin embargo, se decidió realizar un diseño orientado a objetos que se fundamente sobre las reglas vistas en la materia para obtener una mejor implementación del sistema. Es por esto que se decidió no tener un objeto muy poco cohesivo, como sería esta elección multitarea, sino que se dividieron las responsabilidades en diferentes objetos que representen diferentes entes de la realidad. No es lo mismo querer postularse a candidato que querer emitir un voto, por lo que nos pareció interesante tener estas funcionalidades bien desambiguadas. Además, esta decisión tambien gana en otro tema muy importante para un buen diseño que es el bajo acomplamiento. Al separar las funcionalidades, el sistema es mucho más tolerante a lo que puede suceder si solo una de estas funcionalidades cambia su comportamiento. En una elección próxima, la postulación a candidatos podría ser completamente diferente y esto no afectar a todo lo referente a la emisión de un voto, o al sistema de logging.

- Usuarios reificados. Charlando con los diferentes docentes de la materia, nos resulto interesante reificar el concepto de usuario del sistema. A priori un usuario podía ser simplemente un identificador, pero nos parecio adecuado reificar este concepto para tener información que de otra forma correspondería a un sistema externo. Reificando el usuario y teniendo, por ejemplo en el caso del Alumno, la cantidad de materias, lo que podemos hacer es tener una sola interacción con un sistema externo al comienzo de un acto electoral y de esa forma ya tener la información necesaria en los objetos pertinentes, sin hacer más complejo el diseño. De hecho, nos parece que también es una buena forma de mostrar la principal característica de SCRUM, que es ser una metodología iterativa-incremental; al modelarlo de esta forma, se trivializa un poco la interacción con los sistemas externos, dejando esta funcionalidad para un sprint posterior. Por el momento, el modelo soporta el hecho de que al comenzar un acto electoral se cargan los padrones de los usuarios y con esto se obtiene la información deseada. Esta forma de ingresar la información de los usuarios es la funcionalidad que se podría incrementar en una iteración posterior.

- Utilización de interfaces. En variadas ocasiones se nos presento el problema de como modelar la interacción entre todos los objetos del sistema y el usuario final que se encuentra detrás de la pantalla. Existen muchos mensajes que van a ser *lanzados* según las acciones de un usuario del software en el mundo real. De esta forma, parecería que la interfaz gráfica tiene muchas responsabilidades porque necesita *hablar* con muchos objetos. Por ejemplo para poder votar, parecería que el usuario del software tiene que hablar directamente con una urna, cuando no queremos eso ya que no es deseado que se pueda interactuar con el modelo interno del sistema. Por otro lado, tampoco es deseado que la interfaz gráfica de un software, como podría ser la pagina web de este sistema, tenga más lógica que simplemente renderizar lo que se le pide.

  Es por esto que se decidio implementar una serie de interfaces para poder interactuar con el sistema de forma controlada. De esta manera, lo que sucede es que cuando un usuario del mundo real se loguea al sistema, la interfaz gráfica queda *pegada* a una interfaz de uso del sistema, que será diferente dependiendo en la etapa que se encuentre la elección. Por ejemplo, si un usuario se

loguea cuando la votación no esta abierta, sino que se estan postulando los candidatos, la interfaz gráfica solo tendrá comunicación con la interfazParaPostulación, la cual solo permitirá realizar acciones de postulación.

También se utilizó una interfaz para el logging que será explicada cuando se explaye esta funcionalidad en particular.

- Subclasificación de las interfaces.

  Luego de resolver la necesidad de utilizar interfaces para poder interactuar entre el mundo real y el sistema externo, se vió que se necesitaban diferentes interfaces según la época del acto electoral que transcurra, de modo que el sistema tenga un bajo acomplamiento y no haya un solo objeto manejando todo. Dado que se tienen diferentes interfaces, pero que todas representan la misma idea de ser la *cara* del sistema frente al usuario real, nos pareció pertinente realizar una subclasificación de las interfaces ya que esta idea se condice con el mundo real, en donde esta desambiguada la responsabilidad de efectuar la votación, la de postularse a candidato y las demás.

6.1. Logging

En esta sección se presenta todo lo referente al loggin al sistema. Al comenzar el diseño del tp, esta cuestión no fue tomada como esencial ya que no era una de las funcionalidades básicas a tener en cuenta en este sprint. Sin embargo, a medida que se fue desarrollando el diseño de todo el sistema, fue emergiendo la necesidad de explicar el modulo de logging, ya que resuelve cuestiones importantes para las demás funcionalidades del sistema.

A continuación se presenta la parte del diagrama de clases que se refiere al logging para poder explicar de forma más detallada las responsabilidades de cada clase y las reglas de diseño que nos llevaron a realizar el mismo.

ACA va el diagrama de clases.

6.1.1. interfazLogueador

La interfaz del logueador es un objeto que representa cual va a ser la interacción del usuario real cuando quiera loguearse al sistema. Como se verá en la explicación del logueador, este tiene ciertas funcionalidades e información que no queremos que sea accesible ni manejable desde el exterior. Es por esto que se decidió utilizar una interfaz de logueador con la siguiente responsabilidad.

- login: password:. Este mensaje contiene la única responsabilidad de la interfaz de logueador, que es permitirle al usuario real loguearse en el sistema.

La principal idea de esta interfaz se basa en las reglas de diseño vistas durante las prácticas de la materia que nos indica que es una buena decisión poner una interfaz para poder interactuar con cosas interiores al modelo de una forma controlada. De esta forma un usuario real solamente podrá acceder a la funcionalidad de logging y no podrá acceder a otra información interesante que pueda tener el modelo interno de datos.

6.1.2. Logueador

El logueador es un objeto que tendra como responsabilidad manejar todo lo pertinente al ingreso al sistema de una persona. Si bien su responsabilidad principal es el mensaje de logging, maneja otras informaciones necesarias para el loggin como saber en que etapa de las elecciones estamos. Estas son cosas que no queremos que el usuario real pueda acceder, por lo cual justifica la necesidad de una interfaz para interactuar con este objeto.

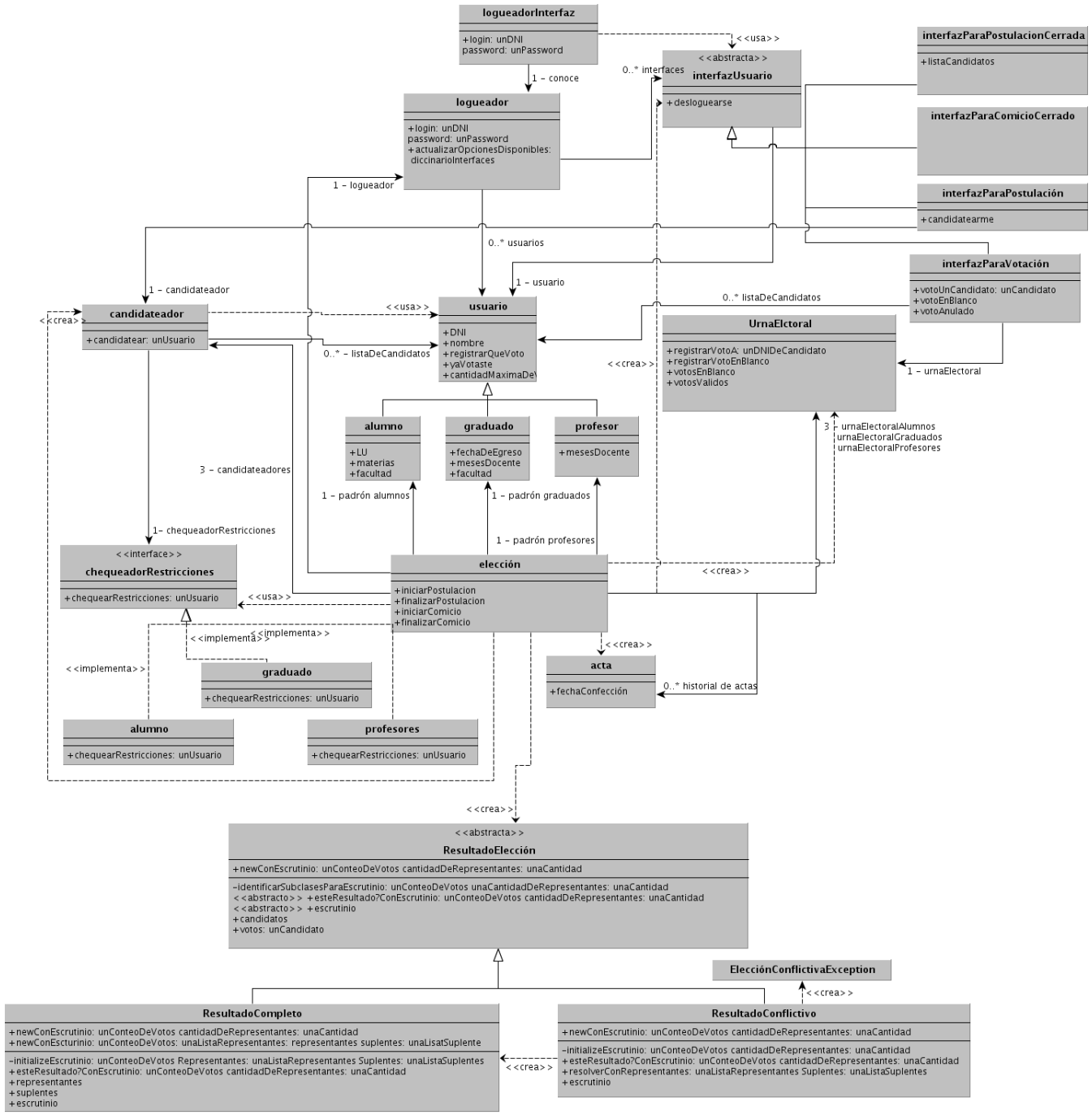
El logueador funciona de la siguiente manera. El objeto sabe que interfaz le corresponde a cada usuario, por lo que el mismo tiene una representación interna de las asociaciones entre los usuarios del sistema y las interfaz de usuario. Esta representación se va modificando dependiendo el estado de las elecciones actuales (postulación, idle, votación, etc). Lo que hace el logueador es entonces, cuando

loguea a un usuario, devolver una interfaz de usuario según la representación que tenga de la asociación entre el usuario que quiere loguear y la interfaz que le corresponde. Es decir, si la elección está en época de votación entonces los usuarios del sistema estaran asociados a interfaces de usuario que en realidad son interfaces para votación. De esta forma, lo que hace el logueador es simplemente devolver la interfaz que esta asociada para que se puede interactuar con el sistema, por ejemplo para votar, mediante esta interfaz que es nuestra fachada para interactuar con el sistema interno.

El logueador resuelve esta situación mediante los siguientes mensajes:

- login: password:. Es el mensaje esencial de un logueador que es justamente su función principal, la de loguear. Este mensaje va a ser utilizado desde la interfaz de logueador que es la única manera que tiene un usuario real para hablar.

- actualizarOpcionesDisponibles:. Este mensaje es el que permite asociar a los usuarios con las diferentes interfaces disponibles para interactuar con el sistema. De esta manera, los usuarios siempre tendrán una interfaz de usuario asociada, pero al utilizar este mensaje, esas interfaces podrán cambiar, asociandose con la interfaz necesaría para la época de la elección pertinente. Entonces, a partir del uso de este mensaje, cuando los usuarios se logueen, podran interactuar con VOX mediante la interfaz que les corresponda con ese período de las elecciones. Entonces si utilizamos este mensaje para asociar a los usuarios con una interfaz de postulación, lo que estaremos logrando es que el usuario cuando se loguee solo puede interactuar con el sistema mediante una interfaz de postulación que solo tendrá como opción de funcionalidad posible el hecho de candidatearse; ya que por la época de las elecciones en las que se encuentra el sistema, no se debería poner hacer otra cosa.



6.1.3. elección

En esta funcionalidad en particular, la de loguearse, las responsabilidades de esta clase se basan en manejar los tiempos de las elecciones para saber que se puede hacer con el sistema en este momento. La clase elección es el principal punto de interacción para un administrador del sistema. Lo cual probablemente induciría a tener una interfaz para administrador que será adherida al modelo, en caso de necesitarla, en otra iteración del presente desarrollo.

Luego, este objeto es responsable de indicar en que etapa de las elecciones esta el sistema, mediante la asignación de las diferentes interfaces a los usuarios. Los mensajes para cumplir con estas responsabilidades son:

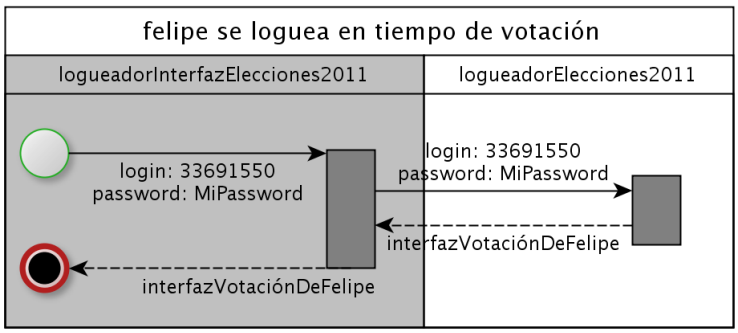
- iniciarPostulacion:. Este mensaje se encarga de hablar con el logueador para que ahora todos los usuarios tengan asociada una interfaz para postulación.

- cerrarPostulacion:. Este mensaje sirve para que cuando falten 14 dias para la elección la postula-

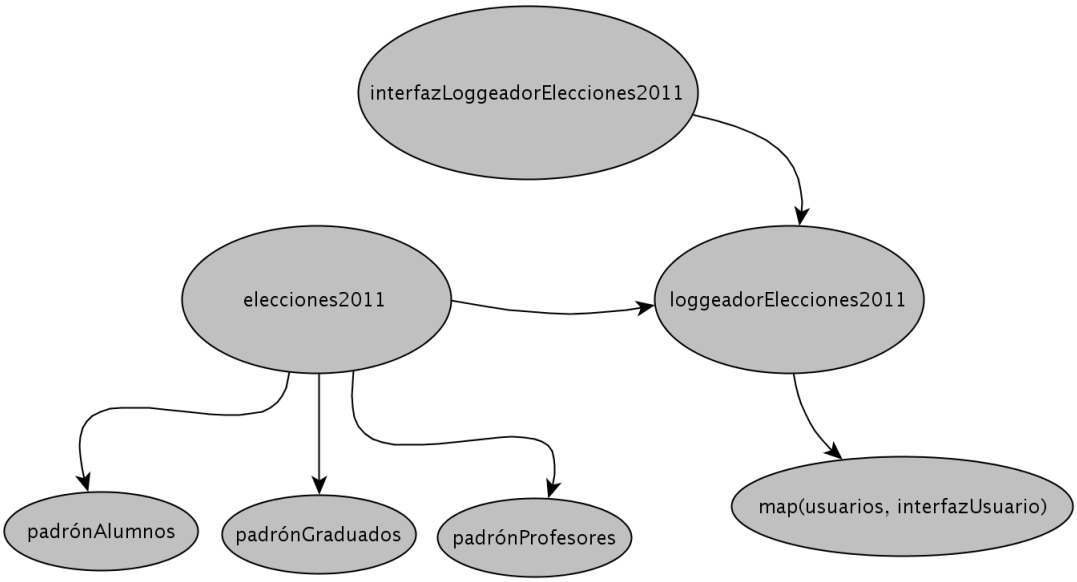
ción se cierre y ahora los usuarios reales no puedan interactuar con la interfaz para postulación, sino que solo pueden ver los candidatos.

- iniciarComicio: Este mensaje sirve para asociar a los usuarios con las interfaces de votación de manera que sea posible interactuar con el sistema para poder emitir un voto.
- FinalizarComicio: Con este mensaje se da por cerrada una elección y esto influye en el sistema de loggin por que ahora los usuarios reales solo podrán interactuar con una interfaz que lo único que les deja hacer es ver los resultados de la elección.

A continuación, se presenta un diagrama de secuencia que sirve para entender de forma dinámica la interacción entre los diferentes objetos en una traza particular para la funcionalidad de logueo.



Por último, se presenta un diagrama de objetos, para poder identificar las colaboraciones entres los diferentes objetos del sistema en un momento en particular.



6.2. Postulación a candidato

En esta sección se presenta la funcionalidad particular de postularse para ser candidato en las elecciones.

Para esto, se reproduce el subconjunto de clases del diagrama original, que corresponden a esta sección en particular.

6.2.1. candidateador

La idea del candidateador es tener un objeto que reifique el concepto que se tendría en una votación sin un sistema informático, en el cual un potencial candidato iría hacia una persona dedicada a la tarea de la postulación para postularse. Esta persona sería la encargada de, por un lado, mirar que el aspirante no aparezca ya en la lista de candidatos, de forma que no pueda haber un candidato postulado dos veces. Luego, esta persona sería también la encargada de utilizar algún sistema externo, o algún catalogo de legajos, para fijarse si el potencial candidato cumple los requisitos necesarios. Nos parece que la tarea de chequear las restricciones es una responsabilidad grande como para asignarsela a un objeto dedicado solamente a eso. Es por esto que en este punto, el candidateador solo chequeará las restricciones colaborando con otro objeto que se encargará del chequeo propiamente dicho.

Luego, para resolver las responsabilidades de una forma bien cohesiva, se tiene un único mensaje:

- candidatear:. Este mensaje es la única responsabilidad que tiene el objeto, recibe un usuario a candidatear y se fija si ya pertenece a su lista interna. Además, en la implementación de este mensaje, el objeto candidateador colabora con el Chequeador para que este le diga si el postulante cumple con los requisitos. Dependiendo el caso del postulante, el candidateador lo sumará a la lista de candidatos o no.

6.2.2. interfazParaPostulación

Esta interfaz tiene un comportamiento similiar a todas las demás interfaces de usuario. Lo que hace es dar una puerta de entrada al sistema para poder interactuar con el subsistema de postulación, pero sin tener acceso a mensajes o informaciones internas que no queremos que un usuario real pueda acceder. Si el usuario real pudiese interactuar directamente con el candidateador, podría eventualmente tener acceso a la lista de candidatos, situación que no queremos que ocurra. Es por esto que la única responsabilidad de la interfaz es tener un mensaje para candidatear a un usuario, y se encarga de enviar esta petición al candidateador real.

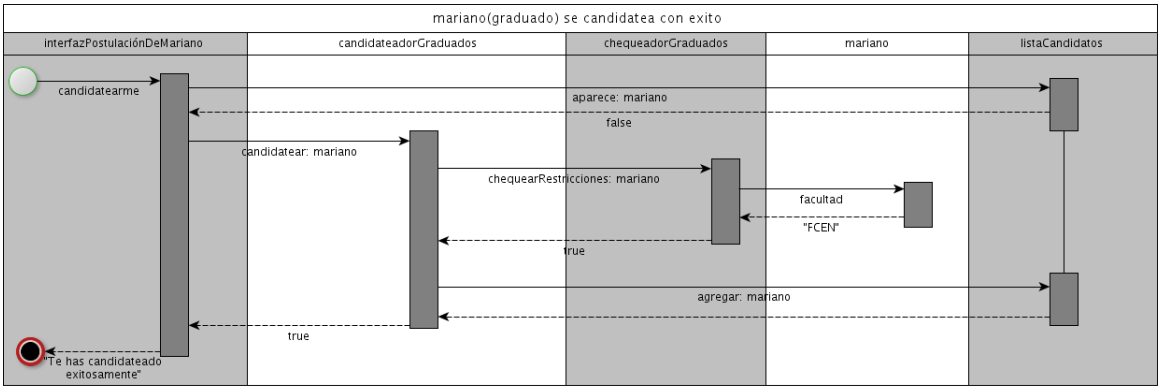
6.2.3. Chequeador de Restricciones

El problema de saber si un postulante cumple con los requisitos para ser candidato no es una responsabilidad menor, sobretodo por el hecho de que queremos darle extensibilidad a esta parte del diseño, permitiendo que cambien los reglamentos por los cuales se chequean los requisitos.

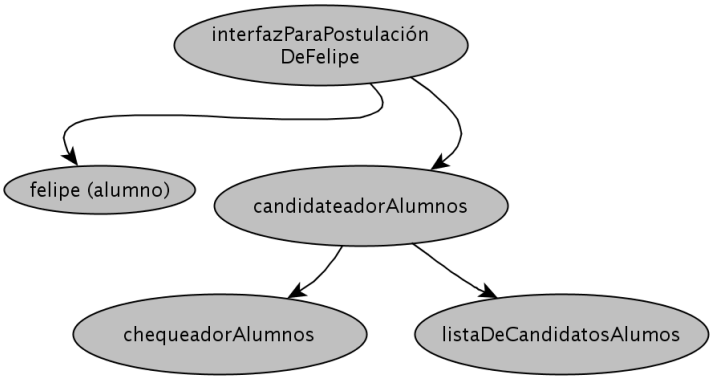
Es por esto que existe el objeto chequeador de restricciones para desacoplar esta responsabilidad del candidateador. La clase chequeadorDeRestricciones además se encuentra subclasificada en chequeador de alumnos, chequeador de graduados y chequeador de profesores. De esta manera, los diferentes candidateadores conoceran diferentes chequeadores de forma de poder delegar las responsabilidad del chequeo en estos.

Cada tipo de chequeador diferente, chequea mediante el reglamento propio a cada claustro, de forma que se puede desacoplar el chequeo de un graduado del de un alumno o un profesor.

A continuación se presenta la traza correspondiente a un graduado que se quiere candidatear mediante un diagrama de secuencia. De esta forma, se pueden ver las interacciones que tienen los diferentes objetos cuando un graduado se quiere candidatear.



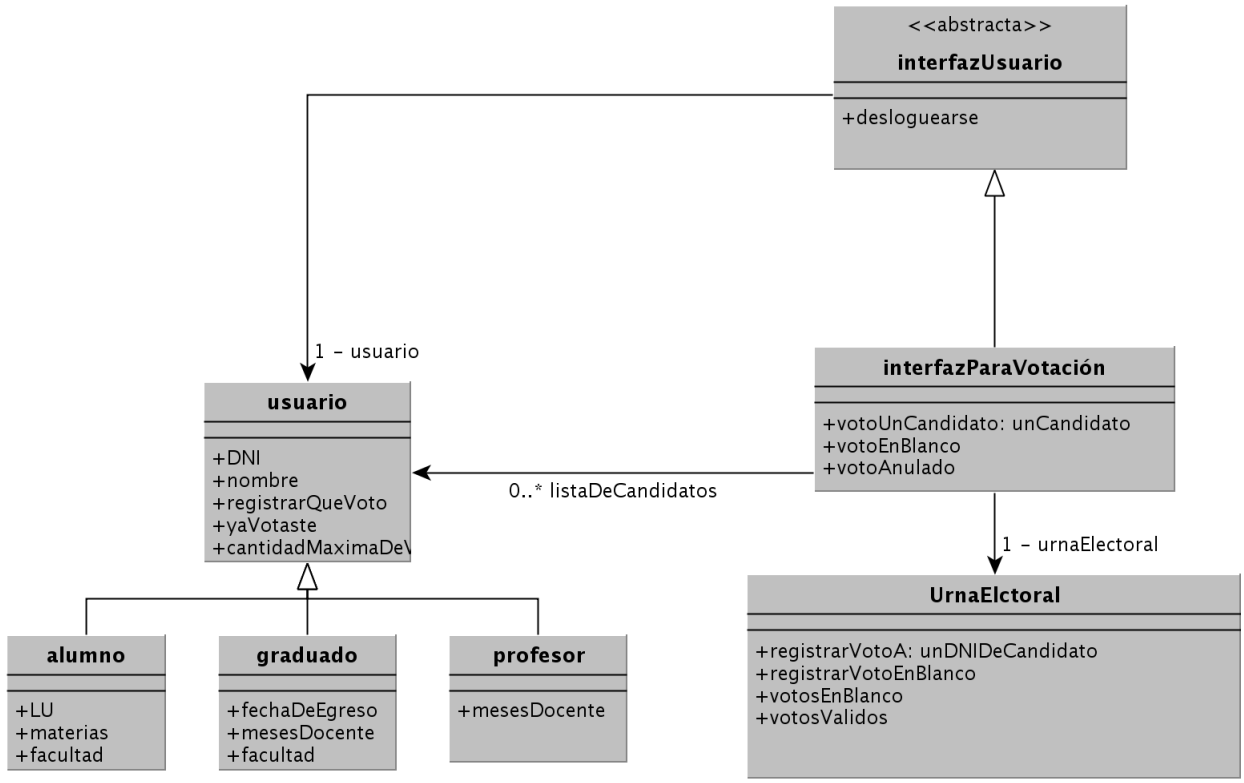
Por último, se presenta un diagrama de objetos para esta funcionalidad.



6.3. Votación

En esta sección se presenta un subconjunto de las clases del diagrama original que son pertinentes al problema de la votación.

A continuación se reproduce dicho subconjunto separado del diagrama original, para luego poder explicitar cada decisión tomada sobre cada una de las clases presentes.



6.3.1. Usuario

El usuario representa a un posible votante o candidato, pero en este caso, solo nos importa el mismo como votante. La entidad del mundo real a la que hace referencia es el votante al momento de iniciarse el proceso electoral. Cualquier usuario debe ser capaz de responder a los mensajes DNI, nombre, registrarQueVoto, yaVotaste. Los mismos se utilizan para:

- DNI: Es un mensaje esencial al usuario, que lo identica como objeto entre los de su misma clase. El mismo hace referencia directa a un usuario del sistema, por lo que también servirá para hacer el recuento de votos. Es decir que los votos en el sistema se dividirán de acuerdo al DNI del candidato al que hacen referencia.
- nombre: Responde al nombre de la persona real que hace referencia el usuario.
- registrarQueVoto: Es un mensaje que permite registrarle al usuario que ya votó para que no pueda volver a votar.
- yaVotaste: Es un mensaje que permite preguntarle al usuario si ya votó. Sirve para cumplir con la restricción de que un usuario no pueda volver a sufragar.
- cantidadMaximaDeVotos: Es un mensaje que responde cuantos postulantes puede votar un usuario.

En cuanto a los mensajes presentandos anteriormente, desde un punto de vista estrictamente paradigmático, no queda claro que sea responsabilidad del usuario saber que ya votó. Sin embargo, se decidió hacerlo de esta forma para reducirle la complejidad al modelo.

Por el mismo motivo se encuentra el mensaje cantidadMaximaDeVotos que, bajo el consejo de Fernando Astesuain, se puso dentro del usuario para que sea simple la votación. Directamente se puede votar una collection de candidatos y chequar cuantos puede votar realmente mediante este mensaje.

En el diagrama, se puede observar que los usuarios estan subclasificados en Alumno, Graduado y Profesor. Esta subclasificación no es muy relevante a esta parte del problema, ya que concierne a las

restricciones para las postulaciones de candidato. Como se verá luego, la interfaz de votación es la que sabe si se trata de un alumno o graduado, o si se trata de un profesor, para saber si puede votar una o dos veces.

Esto lo hicimos de esta forma para seguir una de las principales reglas de diseño en cuanto a que los objetos sean cohesivos. Buscamos tener un usuario que no sepa hacer cosas como votar, o candidatearse o tener otras responsabilidades; sino que pueda responder mensajes muy básicos con pocas responsabilidades.

6.3.2. urnaElectoral

La urnaElectoral reifica el concepto de urna que se tendría en el dominio del problema. La urna es la encargada de llevar los votos de cada una de las votaciones que haya. Una primer idea había sido subclasificar la urna dependiendo del claustro a la que pertenezca. Sin embargo, con la busqueda de tener objetos bien cohesivos, nos dimos cuenta que realmente no importa a que claustro pertenezca la urna, sino que simplemente debe guardar los votos de la elección para la que fue asignada. Esto quiere decir que el sistema problablemente tenga más de una urna, que van a ser una para Alumnos, una para Graduados y una para Profesores; pero estas urnas van a hacer equivalentes, son todas instancias de la misma clase porque las responsabilidades que tienen son las de ser una urna independientes del claustro. Dicho esto, las responsabilidades de una urna son:

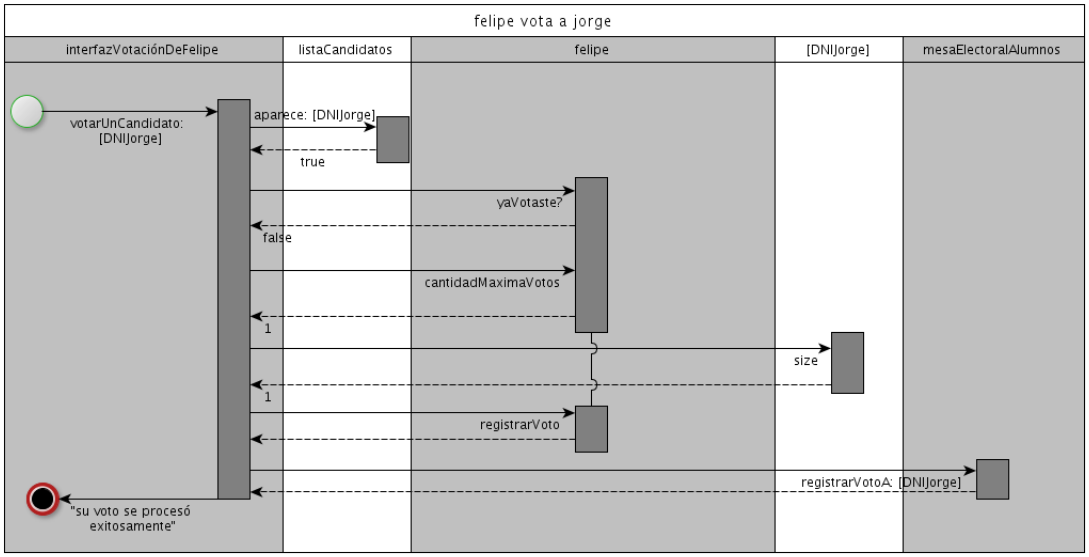
- registrarVotoA:, Este mensaje lo que hace es justamente *meter* un voto en la urna. Dado que lo único que queremos es registrar el voto, la urna solo necesita saber a quien se esta votando.
- registrarVotoEnBlanco. Este mensaje deja registrar un voto en blanco. Nos parece que es una buena decisión de diseño desambiguar el voto en blanco respecto del voto válido, para tener conceptos diferentes reificados de forma diferente y que un voto en blanco no sea simplemente un voto a nil de un voto válido.
- votosEnBlanco. Es el mensaje que representaría abrir la urna y poder contar los votos en blanco que hubo.
- votosValidos. Es un mensaje que nos devuelve una lista de candidatos junto con la cantidad de votos que tuvieron durante la elección.

6.3.3. interfazParaVotación

La interfaz para votación es el objeto que principalmente maneja el sufragio de un usuario. Representa lo que en el mundo real sería la autoridad de mesa cuando el votante va a emitir su voto. Desde este punto de vista, la responsabilidad de los objetos de esta clase es manejar el momento del voto en sí. Este objeto intenta ser lo más cohesivo posible responsabilizandose solamente por el acto de la votación, mientras que las demás funcionalidades quedan como responsabilidad de otros objetos. Para lograr encargarse de estas cosas, las instancias de esta clase saben responder los siguientes mensajes:

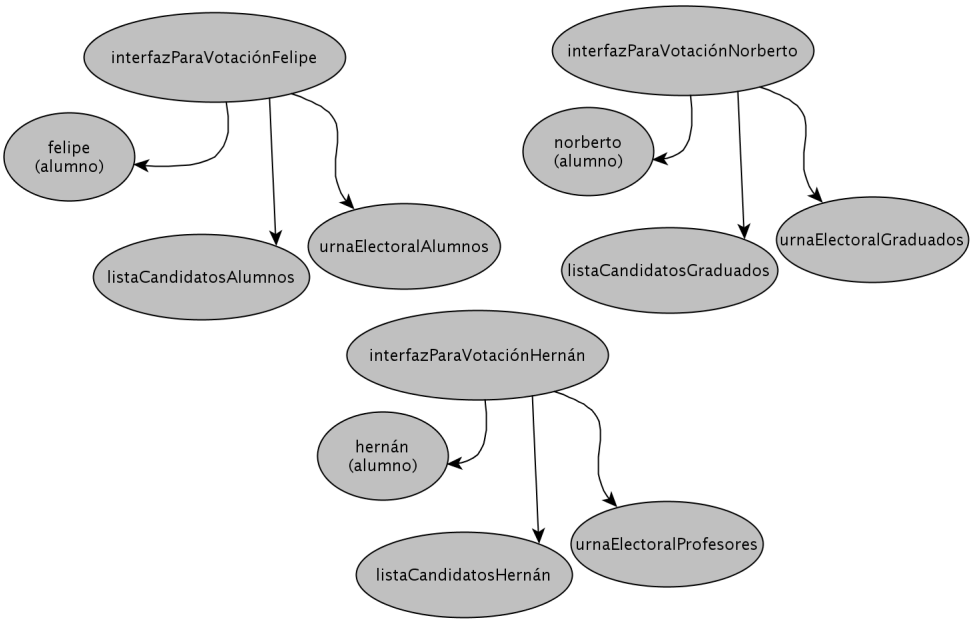
- votarCandidatos:. Este mensaje toma una colección de candidatos a votar y trata de emitir el voto. Al tratar de emitir el voto, se chequea que el usuario no haya votado, y se chequea que la cantidad de gente que esta queriendo votar sea menor o igual a la máxima cantidad de personas que puede votar. Si se puede emitir el voto, entonces esta interfaz es la responsable de comunicarse con la urna para *introducirle* un voto por cada candidato votado, así como también se responsabiliza de indicar que el usuario ya votó.
- votoEnBlanco. Este mensaje se encuentra para que un usuario tenga la posibilidad de votar en blanco. Nuevamente argumentamos que desde un punto de vista del paradigma de objetos, nos parece bien desambiguar los votos en blanco para que no haya colisión de conceptos del dominio del problema, como nos indica una de las mencionadas reglas de diseño.

Si bien el diagrama de clases, más la explicación del mismo, nos da una idea aproximada de como interactúan los objetos entre sí, presentaremos a continuación un diagrama de secuencia que indica como sería una traza posible de las interacciones dinámicas entre diferentes objetos involucrados en el proceso de votación.



Por último, nos parece interesante mostrar un diagrama de instancias que resalte como sería la relación de conocimiento entre diferentes objetos en el transcurso de una votación.

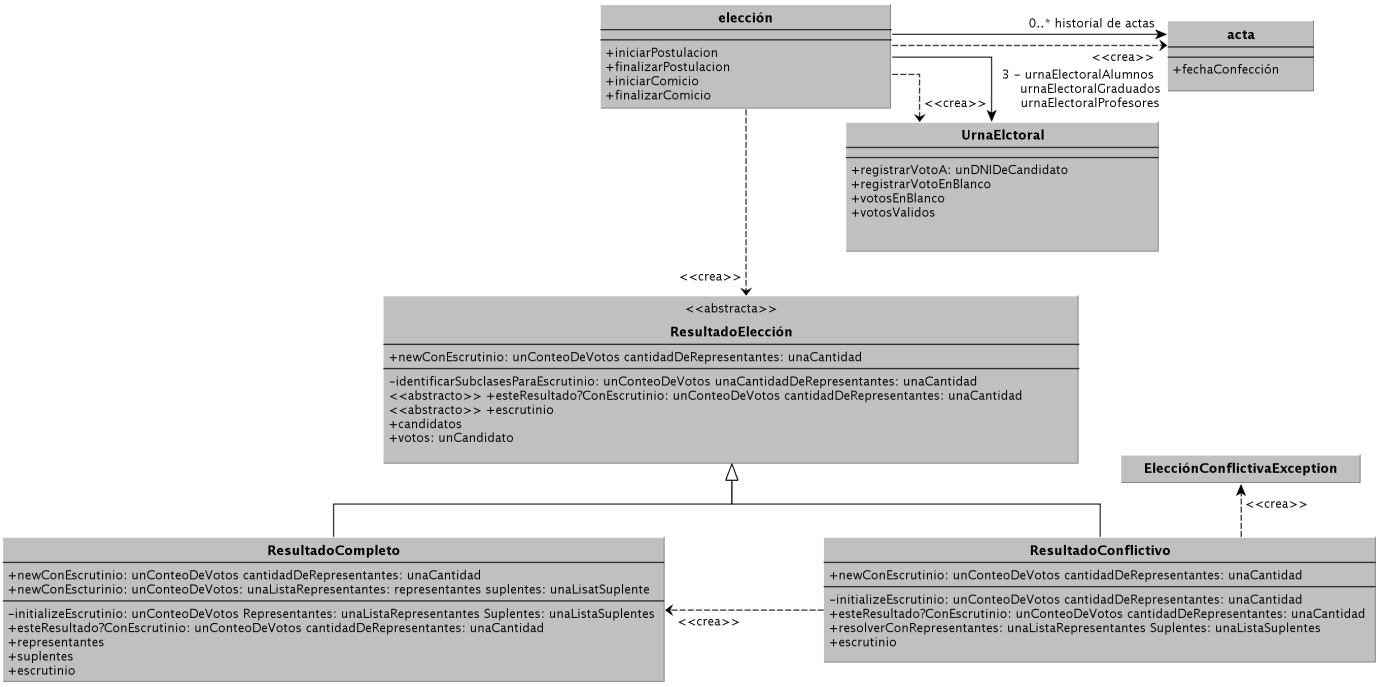
En este caso se presenta una caso donde un Alumno, un Graduado y un Profesor quieren emitir su voto.



6.4. Clausura del acto electoral

En esta sección, se mostrará como se atacó el diseño correspondiente al cierre del acto electoral, en el cual se tienen que resolver algunas situaciones como la generación de actas y la resolución de los conflictos.

A continuación se reproduce el sector del diagrama de clases correspondiente a la clausura del acto electoral.



Lo que se busca en esta simple jerarquía es reificar los distintos posibles resultados de una votación correspondiente a un claustro. Uno de los objetivos al diseñar este módulo fue permitir extender fácilmente la lógica de Vox agregándo resolución automática de conflictos a las funcionalidades provistas.

La implementación entregada como prueba de concepto permite agregar resolución a un tipo de conflicto en particular sin modificar en absoluto el código existente. Esto se puede realizar agregando una subclase de ResultadoEleccion que devuelva una instancia de True al mensaje esteResultado?conEscrutinio: cantidadDeRepresentantes: . La manera en la que lo implementamos fue utilizando una ligera modificación del patrón Factory Method, cuya implementación fue mostrada en las clases teóricas como ejemplo de un idiom de Smalltalk.

Por mencionar un ejemplo, se podrían resolver automáticamente los conflictos de empate sin modificar el comportamiento en los casos de falta de candidatos, siendo conflictos de empate en los que ocurre un empate de votos entre dos candidatos y no es posible distinguir cuál de los dos debe ser representante y cuál suplente.

6.4.1. ResultadoEleccion

La clase ResultadoEleccion representa el resultado de una elección y como tal no existe una instancia concreta del mismo, es por eso que resulta una clase abstracta.

6.4.2. ResultadoCompleto

La clase ResultadoCompleto por su parte representa el resultado de una elección en la que se pudo determinar sin ambigüedades cuáles serán los representantes y cuáles los suplentes, ya sea debido a que la misma no existió desde un principio o porque la misma fue resuelta ya sea por la junta electoral o eventualmente por algún algoritmo incorporado a Vox.

6.4.3. ResultadoConflictivo

Por último, la clase ResultadoConflictivo representa el resultado de una elección a partir del cual no puede automáticamente deducirse qué candidatos deberían ser representantes y cuáles suplentes. Esta última admite la resolución manual del conflicto ingresando una lista de representantes y de suplentes. No se aplican restricciones sobre la lista de representantes y suplentes, excepto que la longitud de la lista de representantes debe ser exactamente la correspondiente a la elección.

Se está suponiendo que el único eje de cambio en la resolución automática de conflictos es respecto del tipo de conflictos y no de los claustros. La razón para tal suposición es que se cree que tal resolución deberá ser lo más objetiva posible y por lo tanto recurrirá a métodos estocásticos en el caso de empate, por ejemplo, o incluirá automáticamente como representantes incluso a aquellos candidatos que no hayan obtenido el 10 porciento de votos usualmente requerido.

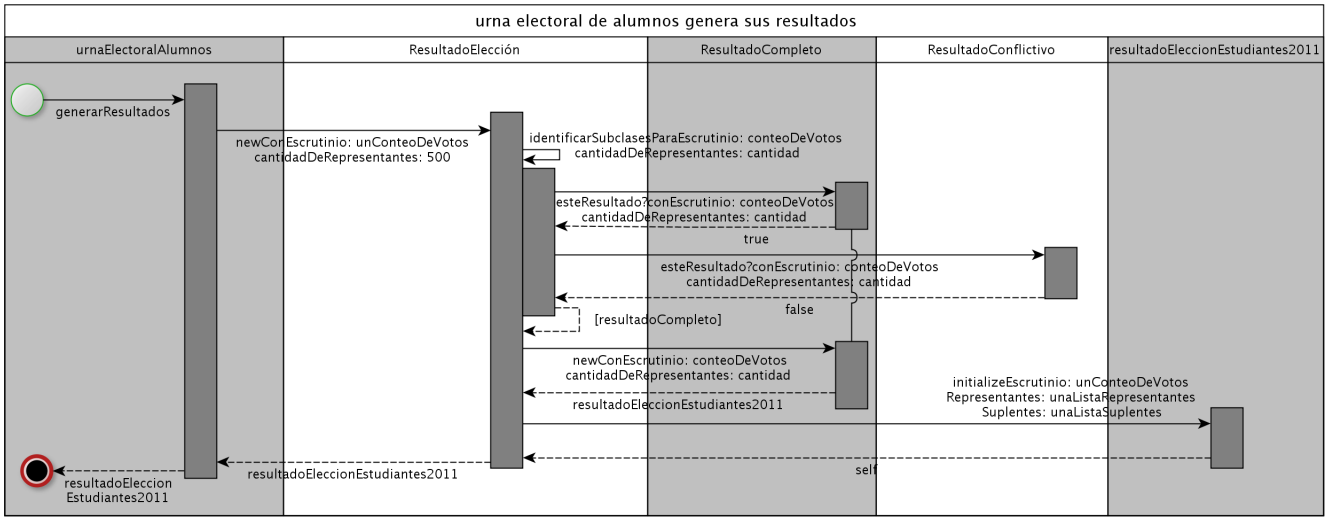
Pese a esta suposición, es importante mencionar que el diseño podría modificarse ligeramente para permitir la inclusión algoritmos de resolución automática de conflictos diferentes para cada claustro. Una buena razón para no hacerlo inicialmente y sacrificar dicha cualidad es que se prefiere priorizar la cohesión de la clase ResultadoEleccion y sus subclases, manteniendo un único eje de cambio.

6.4.4. EleccionConflictivaException

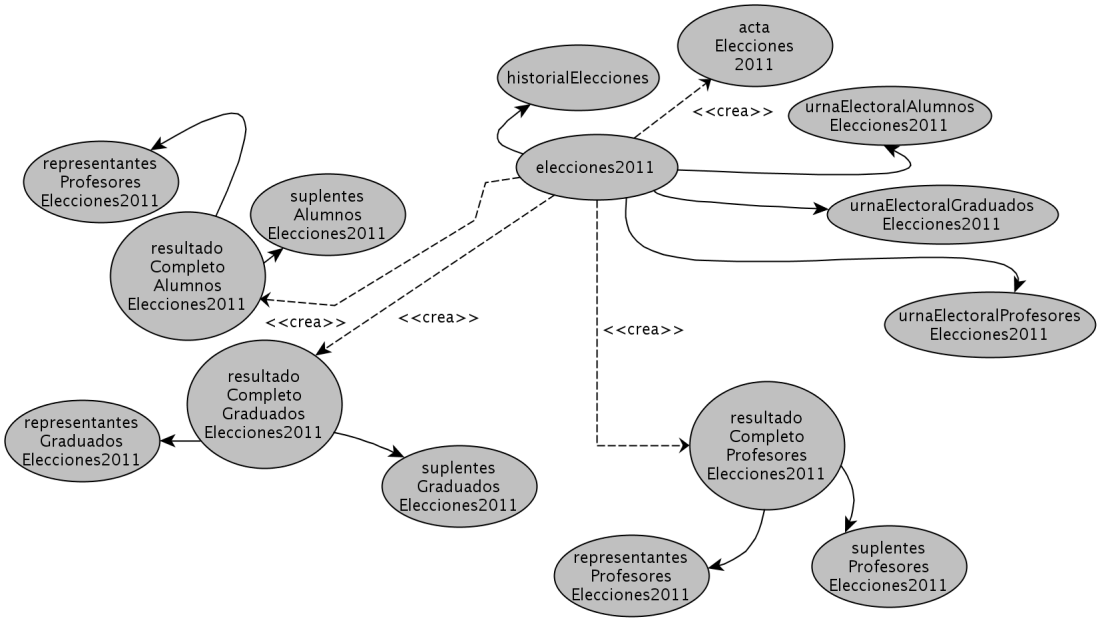
EleccionConflictivaException es una excepción que será lanzada si se manda el mensaje representantes o suplentes a una instancia de la clase ResultadoConflictivo. De esta manera se permite manejar este tipo de situaciones solicitando la intervención de un usuario administrador, que actuará en nombre de la junta electoral, indicando una resolución manual al conflicto surgido. Cualquier instancia de esta clase responde al mensaje parameter con la instancia de ResultadoConflictivo que lanzó la excepción.

Para la mejor comprensión de la implementación entregada es importante comentar que la semántica de ResultadoEleccion >> newConEscrutinio: cantidadDeRepresentantes: es crear una instancia de una de sus subclases cuando dicha subclase es la única que responde afirmativamente al mensaje esteResultado?conEscrutinio: cantidadDeRepresentantes:. De no existir una subclase que responda afirmativamente a dicho mensaje se retorna una instancia de ResultadoConflictivo. Por esta razón ResultadoConflictivo responde siempre negativamente a dicho mensaje. Si existieran varias subclases de ResultadoEleccion que fueran compatibles con los parámetros, se lanza otra excepción, que no es incluída en los diagramas de clases porque consideramos que no es esencial al modelo, sino accidental y debida a la implementación de la prueba de concepto.

El siguiente diagrama de secuencia muestra las colaboraciones al darse por finalizados los comicios. En este caso no surgieron ambigüedades en cuanto a los resultados.



Por último se presenta un diagrama de objetos que representan las relaciones de conocimiento entre los diferentes objetos.


acta
Elecciones
2011
historialElecciones
<<crea>>
urnaElectoralAlumnos
Elecciones2011
representantes
Profesores
Elecciones2011
suplentes
Alumnos
Elecciones2011
elecciones2011
urnaElectoralGraduados
Elecciones2011
resultado
Completo
Alumnos
Elecciones2011
urnaElectoralProfesores
Elecciones2011
<<crea>>
<<crea>>
<<crea>>
resultado
Completo
Graduados
Elecciones2011
representantes
Graduados
Elecciones2011
suplentes
Graduados
Elecciones2011
resultado
Completo
Profesores
Elecciones2011
representantes
Profesores
Elecciones2011
suplentes
Profesores
Elecciones2011

7. Conclusiones

Luego de realizar el tp, el grupo llegó a varias conclusiones que son presentadas a continuación.

7.1. SCRUM

La metodología de scrum parece ser una muy buena idea para implementar proyectos de magnitud acotada, en los cuales resulta interesante ir teniendo versiones del sistema con funcionalidades pequeñas, que luego son incrementadas en cada iteración del proceso.

Sin embargo, nos parece que las virtudes de esta metodología no emergieron durante la realización del presente trabajo práctico, no por un problema de la metodología, sino por una incompatibilidad de la misma con la naturaleza de un trabajo práctico. Por nombrar un ejemplo, es muy dificil querer realizar un sprint en donde las personas que forman parte del mismo no manejan los mismos horarios para dedicarse al mismo, y donde además esta dedicación fluctúa considerablemente dependiendo de otras actividades.

7.2. Diseño orientado a objetos

Como conclusión sobre el diseño de objetos, podemos destacar que nos parece que hacer un buen diseño tiene varias ventajas para modelar la realidad de una forma correcta. Entre las ventajas claras se puede mencionar la simplicidad de la implementación una vez que el diseño esta bien definido, así como también se destaca el hecho de que hacer un buen diseño nos ayuda a que el sistema tenga extensibilidad donde nosotros queramos, ya que justamente se diseño con ese preconcepto.

Por otro lado, nos parece que hacer un buen diseño de objetos no es algo facil de aprender y, auguramos, debe ser una habilidad que se obtiene más de la práctica misma que de la adquisición de nuevos conceptos. La desventaja que le vemos a este tipo de diseños proviene justamente de esta dificultad para generar un buen diseño con poca práctica, ya que al no poder discernir entre las bondades de diferentes diseños, se utilizó demasiado tiempo iterando sobre el diseño mismo, lo cual funcionaba como barrera para todas las demás tareas para realizar en el tp.

7.3. Diagramas UML

En varias oportunidades durante el tp realizamos diagramas de clase, objetos y secuencia, parecidos a los estandarizados por el UML. Esto nos generó algunos problemas debido a que para ninguno de ellos teníamos una formación sólida de como elaborarlos. Sin embargo, nos parece que esto ayuda al dinamismo de la creación de los mismos para no centrar la complejidad en la notación, y si centrarla en el diseño mismo.