Sistemas Distribuidos

Laboratorio Parte 2

Noelia Díaz Alejo Alejo

Organización del repositorio

config	Presenta las configuraciones de Authentication, Icebox y IceStorm
icedrive_authentication	Clases Authentication, Query, Responses, Discovery, etc
client_testing	Presenta los clientes de pruebas
unit_testing	Test unitarios (solo del primer entregable)

Anunciamientos

Primero, obtenemos el topic de IceStorm y si no existe creamos uno.

```
properties = self.communicator().getProperties()
topic_name = properties.getProperty("DiscoveryTopic")
topic_manager = IceStorm.TopicManagerPrx.checkedCast(self.communicator().propertyToProxy("IceStorm.TopicManager.Proxy"))
try:
    topic = topic_manager.retrieve(topic_name)
except IceStorm.NoSuchTopic:
    topic = topic_manager.create(topic_name)

adapter = self.communicator().createObjectAdapter("AuthenticationAdapter")
adapter.activate()
```

Para publicar eventos es necesario utilizar el publisher (debe ser un proxy de tipo Discovery). Él será el encargado de anunciar mi servicio Authentication.

En app.py: Se convierte el publisher al tipo correcto (proxy de tipo Discovery).

```
discoveryProxy = IceDrive.DiscoveryPrx.uncheckedCast(topic.getPublisher())
```

Para escuchar eventos es necesaria una instancia de Discovery que este suscrita al publisher. Debe informar de todos los servicios que se descubren y guardarlos.

Discovery se encarga de guardar los proxys de cada servicio (Authentication, Directory, Blob) y mostrarlos. Para ello, hago uso de conjuntos. Un conjunto es una colección de elementos únicos y sin un orden.

En app.py:

1. Creo una instancia de Discovery.
2. La añado al adaptador.
3. Se suscribe al topic para recibir eventos.

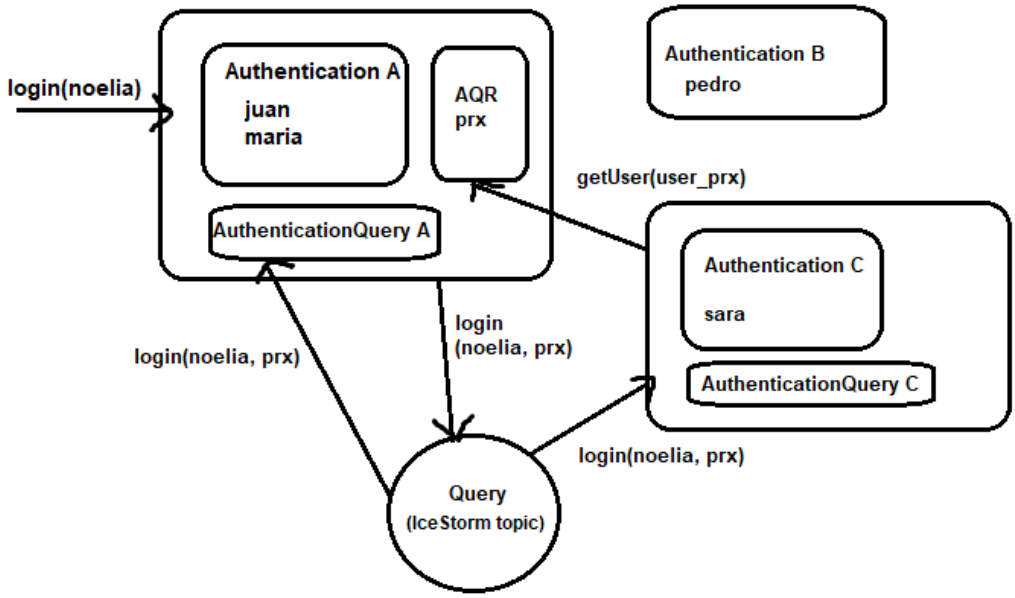
```
discovery_servant = Discovery()
discovery_proxy = adapter.addWithUUID(discovery_servant)
discovery_topic = topic.subscribeAndGetPublisher({}, discovery_proxy)
discoveryProxy = IceDrive.DiscoveryPrx.uncheckedCast(topic.getPublisher())
```

Cada 5 segundos se deben enviar anunciamientos. Para ello, utilizo un hilo.

```
Thread(target=announce, args=(discoveryProxy, authentication_proxy), daemon=True).start()
```

```
def announce(discovery, authentication):
    while True:
        discovery.announceAuthentication(IceDrive.AuthenticationPrx.uncheckedCast(authentication))
        time.sleep(5)
```

Resolución Diferida



Comportamiento

1. Un usuario intenta logearse con un nombre y contraseña.
2. Si authentication A tiene su nombre y contraseña, se comportará como el entregable 1.
3. Si no los tiene, se produce la respuesta diferida. AuthenticationQuery A pregunta si existe alguna Query que conozca a ese usurario. Si la hay, se manda una Query Responses, si no la hay salta una excepción.

Pruebas: Para las pruebas, me he creado un cliente y dos Authentications con persistencias distintas (cada uno tiene una query distinta).

Query Responses: Debido a que cada método presenta distintas necesidades. He creado dos query responses, una para el login y otra para el newUser.

- **Login**. Se guarda en future el proxy del usuario.
- **NewUser**. Se maneja la excepción correspondiente.

En conclusión, es un mecanismo asíncrono para manejar solicitudes de autenticación.