Escuela de Informática
Universidad Nacional de Costa Rica (UNA)
Proyecto OneFlowStream (OFS)
EIF400 Paradigmas de Programación II-2023. Dr. Carlos Loría-Sáenz

Especificación General de un Proyecto de Programación: Entorno Prototipo para el Lenguaje OneFlowStream (OFS)

Acrónimo de referencia rápido del proyecto para efectos del curso: **OFS_2023**

Introducción

Esta especificación (SPEC) determina los elementos principales para la definición de un proyecto programado sobre paradigmas, lenguajes de programación y compiladores en el marco y alcances del curso EIF400. El producto final esperado es una aplicación Web demostrativa, la que ofrece un entorno prototipo para trabajar con un lenguaje que sirve para expresar computación basada en flujos de datos. Ese lenguaje será denominado OneFlowStream (OFS) para fines interno del proyecto.

Marco de Referencia

El punto de partida es el estudio empírico de lenguajes que expresen computación en forma declarativa pero basados en un modelo de flujo de datos combinado con el paradigma de programación funcional (FP). En tales modelos, la programación toma precisamente la forma de flujos que van transformando los datos en nuevos datos, de forma encadenada, y por demanda, es decir los flujos se consumen solo si son requeridos explícitamente.

Este modelo de computación corresponde a la forma en que los denominados Streams en Java trabajan. El proyecto que se quiere desarrollar es un lenguaje "script" simple que permita expresar ese tipo de flujos de forma tersa, declarativa, a manera de prototipo para estudio, con fines puramente académicos y de aprendizaje.

El proyecto contempla dos etapas de desarrollo separadas pero que deben ligarse al final produciendo un entorno Web en el que sea posible fácilmente editar código en OFS y un API de servicios que permite compilar, ejecutar, almacenar (persistir) código OFS en el servidor.

El objetivo del proyecto es desarrollar el lenguaje OFS, que es uno "casero" definido en el marco del curso, para que permita jugar con ideas basadas en programación basada en flujo de datos y sobre temas de estudio del curso.

La definición específica de OFS (sintaxis y semántica) se dará en su momento durante el desarrollo del curso.

Alcances y Recortes Potenciales

A) Esta especificación asume que el proyecto evoluciona **ágilmente** sin que desde el inicio se conozcan todos los alcances completamente desde el inicio. Es posible, entonces, que estos cambien según el avance del proyecto y condiciones del curso.
B) Es posible que por limitaciones de tiempo o técnicas haya que recortar, modificar o simplificar algunos de los requerimientos que se enumeran a continuación y que dan un marco "amplio". Un área de recorte o simplificación esperable sería el requerimiento de persistencia en el server. Los demás requerimientos se espera que puedan ser cumplidos, siempre limitado el resultado final a las posibilidades y en coherencia con el avance del curso.

Etapas de implementación

Se desarrollará en dos grandes etapas:

Etapa I: Rich Client Dummy Server

En esta se implementa el cliente que permite una funcionalidad y una presentación comparable (más no idéntica, se quiere insistir) a una como Traceur o similares. En presentación y usabilidad hay un gran espacio de libertad para su selección, no es ese el tema del curso. Que sea "apropiada" en presentación y usabilidad, es el mínimo. Pueden usar a conveniencia lo que les facilite el trabajo o mejore su conocimiento.

Como se ve a continuación, es un cliente reactivo que responde al usuario y usa servicios de server. Se quiere usar un framework acorde, como React/NextJS que además está fuertemente asociado con FP (Hooks). Para uniformar la revisión, será obligatorio usar React el framework de cliente pedido y NextJS el framework de server, sobre Node.

Se deja libre la selección del motor de base de datos. Siendo los objetos por almacenar documentos y no siendo una aplicación transaccional, una BD NoSQL sería apropiado, más no obligatorio.

La página, que es una SPA, tiene al menos un menú con una opción de `About`. Podrá tener otras opciones en la segunda etapa. Seleccionando `About` se puede ver quiénes son los autores del proyecto y sus calidades. Puede ser un diálogo pop-up u otra forma de despliegue. Puede haber un gesto para establecer preferencias de la aplicación, pero no es requerido obligatoriamente.

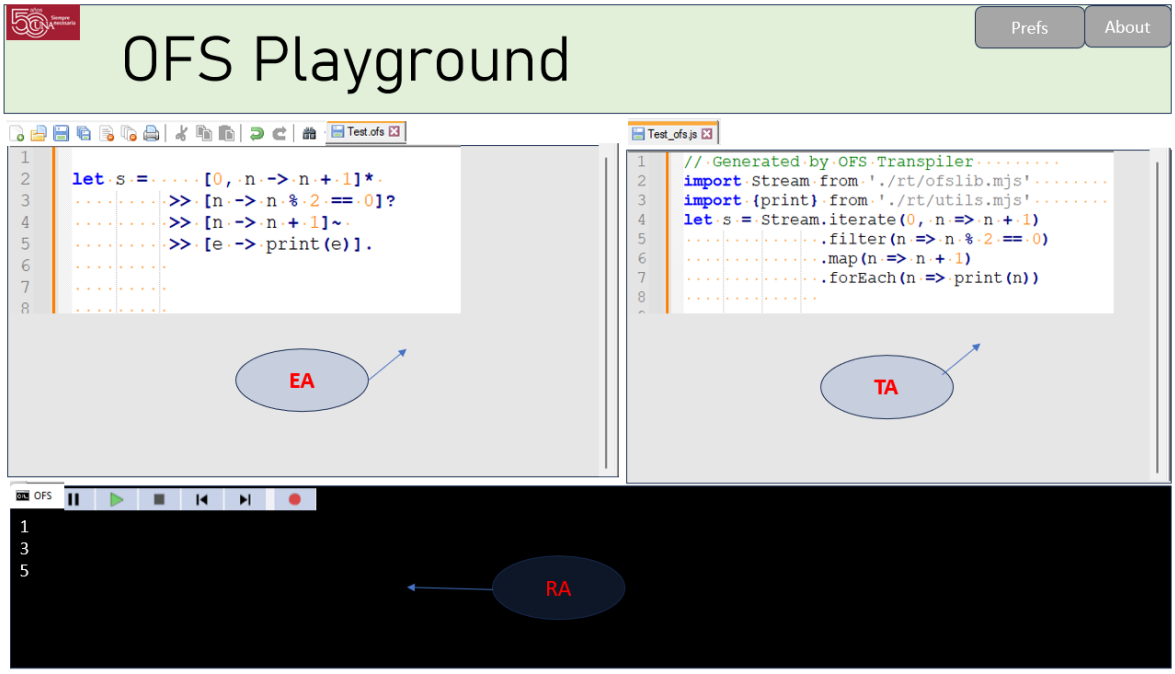

Imagen Fingida de Posible Página

Se asumen tres grandes componentes así: un área textual de edición (`EA`) de código (para un script de `OFS`), una de salida (read only) de transpilación (`TA`) y una de respuesta o salida de ejecución, read-only (`RA`).

En el cliente se puede editar código (en EA), solicitar al server compilar (transpilar), ejecutar y ver resultados (en `TA` de compilación en `RA` de evaluación). Se pueden solicitar traer scripts desde el server (a `EA`). También salvar un script en EA en el server. Para tales funciones, el cliente habla con un `APIREST` de servicios en el server, como los que se perfilan más adelante en este material.

Al invocar funcionalidades puede haber validaciones y por ende diálogos de error o advertencia. Por ejemplo, si se pide evaluar y no hay `EA` y por ende no `TA` asociados. Una opción es añadir una barra de status que pueda indicar el estado sobre los datos visibles. Por ejemplo, si el script ha sido compilado o no. O si se pide evaluar y no ha sido compilado `EA`. O alternativamente desactivar opciones dado el estado de los datos en un momento dado. Ese espacio de decisión es de opción libre.

Sobre el API de servicios: ante el gesto de click en `About`, la página invocará un servicio de `/about`, que responde en `JSON` con un documento que contiene los datos de los miembros del equipo, del curso, del proyecto, semestre y año, escuela y universidad. El cliente muestra esos datos, por ejemplo, en un diálogo.

Ante un gesto de pedir traer un script: en el server hay un servicio `/script/id` donde `id` es el identificador de un supuesto script ya almacenado en el backend. En la primera etapa, eso se podrá simular con un texto que será leído del disco en el filesystem del server. El server responde con ese texto y este se despliega en `EA`.

Ante un gesto de querer compilar lo que está en `EA`: el server tiene un servicio, `/compile`, el que por `POST`, recibe del cliente lo que sea que haya en `EA` (en un JSON adecuado) y el server envía de vuelta lo mismo (en JSON), añadiéndole un timestamp al inicio del texto recibido, en una línea independiente del texto original. Es decir, en esta primera etapa es solo un servicio inicialmente de eco más timestamp. Se va a desplegar para esta etapa ese eco en TA.

Ante el gesto de querer evaluar: el server tiene un servicio /eval que por POST que inicia la evaluación de la versión transpilada del script en EA. Esta interacción puede requerir paginación dada la naturaleza de `OFS` (los flujos pueden ser infinitos). Se determinará más adelante, si se pide esa paginación entre cliente y server, o se retorna un resultado ya completo navegable en el cliente. En esta versión, el resultado de evaluar será en esta etapa de nuevo simulado con un archivo en el filesystem del server.

Etapa II: Richer Client Smart Server

Se mejora el cliente en base a lo aprendido en la Etapa I y se implementan los servicios del server no implementados aún.

En esta etapa el server de Node delegará su (hasta ahora "fake") lógica de negocios en un nuevo servidor privado escrito en Prolog. Por lógica de negocios se entiende compilar scripts de `OFS` y generar la versión en JS del script `OFS`. Otros servicios de entregar scripts, y almacenar en persistencia, se mantienen en Node. El servidor de Prolog es solo accesible desde Node, el cliente no lo conoce. Se debe lograr comunicar Node/Express con ese servidor Prolog sin intervención del cliente.

Los detalles y requerimientos de `OFS` y su compilador se darán a conocer con suficiente tiempo de antelación durante las clases antes de pasar a evaluar esta etapa.

Al final de esta etapa las capas funcionan bien y el diseño e implementación de la etapa anterior resiste el cambio sin grandes modificaciones.

Requerimientos Generales

Se debe desarrollar:

1. Una aplicación Web estilo `SPA` (single-page-application) y un server con un API de servicios web REST-Full que le permite a un usuario de `OFS` crear y procesar ejemplos de código en `OFS`. Los detalles de la sintaxis y su procesamiento serán especificados en su debido momento en adenda a este `SPEC`. La interacción ocurre en el cliente con servicios de apoyo

del backend para la compilación, evaluación y almacenamiento de objetos `de OFS` en el backend. El backend trabaja por un `API` de servicios bien definidos para darle soporte a la página. Los servicios entregan sólo datos no vista; con la excepción de la página `index` o `home` que el backend genera la primera vez que se inicia la aplicación. La arquitectura tiene entonces: a) el cliente, b) un servidor general de backend y persistencia y c) un servidor de lógica de manejo de `OFS`. Esta última sería el equivalente a la lógica de negocios de una aplicación empresarial común. El cliente no conoce este último servidor, usa el servidor general como intermediario (proxy).

2. Un compilador en Prolog que transpile una instancia de un script en `OFS` en un programa en un módulo de `JS`.
3. Un conjunto de módulos de soporte (runtime) que permita la ejecución del código JS generado por el compilador. Este runtime da soporte al procesamiento de streams, sería una biblioteca especial para el proyecto. Esta debe ser propia y original.
4. Los servicios REST necesarios para darle funcionalidad al cliente en cuanto al apoyo a la edición, compilación y ejecución. Para efectos de ejercitar distintos temas del curso, el servicio de compilación en un servidor en `Prolog`. Como se dijo, la comunicación entre los servicios y el cliente no dependerá de que esa capa sea en `Prolog`. La misma podría hipotéticamente ser sustituida en un momento posterior por una equivalente, por ejemplo, en `Java`, sin afectar al cliente.
5. La interacción modular y coherente necesaria para mantener la operación del cliente y la comunicación entre el cliente y los servicios requeridos. El cliente permite salvar y recuperar del backend objetos (scripts de `OFS`). El cliente usa el backend para compilación, evaluación y persistencia.
6. Un proceso de construcción de la aplicación automatizado el que permita construirla de manera expedita y frecuente que incluya unit-testing y empacamiento, **fuera de un IDE**.
7. Casos de prueba unitarios funcionales para el compilador y el proceso de evaluación. Para esto se debe permitir en el cliente cargar `OFS` ya preparados que prueban los servicios de manera más fluida.
8. Una estructura de proyecto estándar en Web apropiada para la comprensión de la aplicación y su build automatizado, empacamiento, distribución y documentación.
9. Una funcionalidad de edición de texto en el cliente. Esta es recomendable basarla mejor en alguna librería de cliente, cuya escogencia queda a libre elección pero que debe poder interactuar con el framework de cliente de manera adecuada a los propósitos (no hacks extaños).
10. El desarrollo de componentes propios que muestren el aprendizaje y aplicación de conceptos del curso en paradigmas, especialmente.

Demos de Apoyo

En su debido momento se podrán entregar, por parte del profesor, scripts demostrativos y se explicarán, para que sirvan como puntos de partida para el desarrollo de este proyecto, su comprensión y como casos de unit-testing a lograr.

Entregable

Como se indicó, se divide el entregable en dos grandes etapas.

- La primera etapa que cubra más el cliente y lo básico estático del backend (eventualmente usando mocking de servicios para probar la interacción entre capas), el proceso de build de la app. Incluye aprendizaje sobre las herramientas principalmente sin entrar en detalles sobre `OFS` aún.
- La segunda etapa itera sobre la anterior de forma que los servicios de backend ya funcionen para el procesamiento de `OFS` según especificaciones que se hayan alcanzado en ese punto del proyecto durante el curso.

Entre esas dos etapas (que serían dos sprints "largos") podrán pedirse sprints cortos de avance, con valor evaluativo.

Criterios de Evaluación

El profesor revisará a su criterio personal y profesional el trabajo tanto en forma como en fondo. Los criterios de evaluación incluyen los señalados abajo. En su momento se dará una guía de puntajes específica con suficiente tiempo antes de la revisión final.

a) Cobertura de los objetivos planteados (el incumplimiento puede hasta anular el proyecto)
b) Uso de las herramientas pedidas a acordadas (el incumplimiento puede hasta anular el proyecto).
c) Cobertura y correctitud de las funcionalidades específicas pedidas (lo contrario puede hasta anular el proyecto).
d) Suficiente calidad del código y uso de FP en los lugares y situaciones que lo ameriten en la adecuada combinación OOP-FP (lo contrario puede anular el proyecto).
e) Calidad de la página principal y las funcionalidades requeridas
f) Respeto a la arquitectura y estilos de programación pedidos, así como herramientas y organización del proyecto (lo contrario puede anular el proyecto)
g) Adecuado ciclo de construcción de la app y su apropiado testing. (lo contrario puede anular el proyecto)
h) Cumplimiento de sprints cortos que se pidan para evaluación parcial de avance
i) Entregable de fácil distribución y ejecución para pruebas fuera del entorno de desarrollo (lo contrario puede anular el proyecto).

Herramientas y Estilos

- Lenguajes: `JS` (`ES>=6`, `FP-OOP`-Reactivo, modular), `Prolog+DCG` (modular)
- El cliente en `React`. Servicios deben ser en API estilo REST. En ES6+.
- Backend: El server general en `Node/NextJS`. El server de lógica de negocios (transpilación) en `Prolog-DCG`.
- La base de datos de escogencia libre.
- Opcional: Usar Typescript.

Valor en la Nota

Este trabajo determina la nota dedicada a proyectos en la carta al estudiante con un valor de un 100% a ese rubro. Cada etapa (entre la I y II mencionadas) aporta un 50% c/u (corresponden a proyecto I y II de la Carta al Estudiante, respectivamente). Se permiten puntos extras en cada etapa, los que pueden hacer la nota mayor que 100 hasta un 25% extra. Solo aplica si se cumple con un 85% al menos de lo pedido como obligatorio. A criterio del profesor.

Igualmente, este trabajo podrá según su calidad ser usado más allá por el docente como un elemento de valoración y aprecio del interés por aprender y ganar el curso y así reflejarlo en el cálculo de la nota final del curso, como una adicionalidad.

Forma de Trabajo y Fechas

Se desarrolla **solamente en los grupos de trabajo formalmente establecidos en el curso** y conocidos por los estudiantes. Se harán sesiones de revisión presenciales. Durante las revisiones todo miembro del grupo debe estar presente y participar activamente y su **participación individual puede afectar la nota individual o grupal**. Las revisiones son en el horario de clase matriculado. Cualquier ausencia debe ser justificada como en una evaluación de examen.

Sobre fechas se plantea tentativamente lo siguiente: La primera etapa **semana 9** y la segunda **semana 17**. Se notificarán la forma y hora de entrega, a definir al menos una semana antes por el profesor. Se podrán pedir avances que afecten la nota final del trabajo.

Para entregar el proyecto se abrirá un drive donde el coordinador subirá el entregable según se le pedirá a cada grupo. El entregable deberá ser liviano, será tal que el profesor lo pueda construir en su máquina. El caso de tener instalado Prolog será manejado como caso especial.

El entregable es independiente de si usan un repositorio como GITHUB, que se recomienda hagan. Pero, el profesor no hará “clone” para tener que revisar el proyecto.

Puntos Extra

Se pueden, a criterio del profesor, conceder puntos extra de hasta un máximo de un 20% adicional sobre la nota del trabajo, siempre que el trabajo cumpla los mínimos pedidos (85%). Los extras deben ser sugeridos por los estudiantes y aceptados de previo por el profesor. Una vez

comprometido un extra se vuelve obligatorio. Se recomienda medir bien los riesgos de las decisiones de extras por las curvas de aprendizaje. Los extras, en ningún caso, podrán cambiar las herramientas obligatorias ni la arquitectura básica ni estilos de programación esperados.