Tesis
Departamento de Computación
F.C.E.F.Q. y N.
U.N.R.C.

recabs

Informe de Diseño

Bessone, *Mariano José*
marjobe@gmail.com

Bringas, *Emanuel César*
emab73@gmail.com

16 de Junio de 2010

Índice

1. Arquitectura

La arquitectura de este proyecto sigue el modelo cliente/servidor, la cual tiene distintas responsabilidades y las mismas se detallan a continuación.

- **Servidor:** Tiene la responsabilidad de iniciar el proceso recursivo, administrar los pasos de recursión que realizan los clientes y llevar el control de los resultados.

- **Cliente:** Encargado de la resolución de un solo functor recursivo, lo cuál puede involucrar pedir ayuda y solicitar el envío de functores intermedios *(Veasé* `RecursiveFunctor`*)* y resultados al servidor.

1.1. Capas

- **Subyacente.** Esta capa debe mantener la arquitectura cliente/servidor teniendo las siguientes responsabilidades.

  - *Servidor:* brindar un servicio de distribución de trabajos a clientes conectados de alguna manera, permitir una interacción con ellos y además informar el resultado final de los trabajos de cada cliente.
  - *Cliente:* recibir un trabajo particular delegando la implementación de este a la capa superior así como soportar interacción con el servidor, lo cuál provee una manera de tratar con trabajos incompletos.

- **Aplicación.** De la misma forma que en la capa anterior, se mantiene la separación cliente/servidor y además una parte común entre ambas, con las siguientes responsabilidades:

  - *Servidor:* iniciar el proceso recursivo declarando el estado del `RecursiveFunctor` inicial.
  - *Common:* definir la generación de hijos de un functor, la serialización del mismo y determinar la reducción de resultados.
  - *Cliente:* iniciar el proceso del cliente y especificar la deserialización de un functor.

2. Interfaces

A continuación se detalla las responsabilidad de cada una de las interfaces empleadas en el diseño de ***recabs***.

2.1. Server side

`RecursionManager`. Es el cerebro de la recursión, quien maneja la pila y quien lleva los resultados parciales de cada paso de la recursión. Provee los siguientes servicios:

- comienzo del proceso recursivo,
- información de resultado final.

Colaborador:`ResultReducer`

`L4ServerApp`. Esta interfaz provee el servicio de obtención del functor inicial del problema. Quien la implemente deberá especificar este functor y serializarlo.

2.2. Common side

`ResultReducer`. Es el encargado de manejar los resultados y su finalidad es llegar a deducir el resultado final. El servicio que provee es el de registrar o acumular un resultado parcial. Los requisitos para el implementador son:

- definir el tipo de resultado que se opera,
- realizar la operación de reducción de un resultado parcial, adecuado al problema.

`RecursiveFunctor`. Es quién representa un estado particular de una función recursiva. Nos brinda el servicio de generación de hijos, es decir, retornar sus estados siguientes en una lista de functores hijos. El implementador deberá, dependiendo en que etapa se encuentre el functor, calcular:

- *Etapa Inductiva:* rellenar la lista de functores hijos.
- *Caso Base:* devolver un resultado del tipo apropiado.

De esta manera la representación de cualquier función recursiva solo pasa por la implementación de esta clase abstracta, especificando el modo de generación de hijos.

`SerializableRecursiveFunctor`. Es un `RecursiveFunctor` para proyectos distribuidos, por ende, se agrega el servicio de serialización de un functor. Requisito: serializar los atributos necesarios y suficientes de un functor en particular.

2.3. Client side

`RecursiveProcessor`. Es el encargado de realizar la ejecución total (o parcial) del functor asignado a un cliente, manipulando los resultados parciales de esta ejecución con la colaboración de un `ResultReducer`.

`DistributableRecursiveProcessor`. Es la versión distribuible de RecursiveProcessor, por lo que se añaden dos primitivas de colaboración requeridas para interactuar con el servidor, las cuales son:

- pedir colaboradores para realizar un trabajo particular,
- despachar unidades de trabajo, para que el servidor las asigne a otros clientes.

El servicio de ejecución que provee esta interfaz está implementado en base al proyecto "Implementación del modelo HPC colaborativo aplicable a problemas de bio-informática" de Intel, adaptado para una arquitectura cliente/servidor.

`DeserializerFunctor`. Provee los servicios de transformación de un paquete serializado en el functor correspondiente. Cabe aclarar que quién la implemente deberá tener en cuenta la implementación del functor para una correcta extracción de datos del paquete.