Trabalho prático[1] #1 de Arquitetura de Software

Introdução

O objetivo deste trabalho é experimentar a aplicação de padrões/estilos arquitectónicos na prática e verificar o seu impacto nas propriedades sistémicas de um programa. Para isso, vamos utilizar o estilo arquitetónico *pipe-and-filter* como exemplo.

O trabalho será orientado para a implementação, de forma a permitir-lhe experimentar uma estratégia de implementação *pipe-and-filter*, a fim de obter uma compreensão mais profunda das questões relacionadas com a implementação de um projeto arquitectónico através do seu código. Note-se que esta não é uma disciplina de programação, e por isso a ênfase desta tarefa é sobre as questões arquiteturais. Simplesmente reescrever todo o código irá indicar uma falta de compreensão dos conceitos arquitectónicos fundamentais.

Objetivos

É-lhe fornecido o código exemplo de um programa que faz uso do estilo *pipe-and-filter*. O domínio de aplicação deste programa é o processamento de sinal, como descrito na próxima secção. A sua tarefa é estender o programa que lhe é fornecido de forma a arquitetar e construir os sistemas especificados pelos requisitos que lhe serão apresentados.

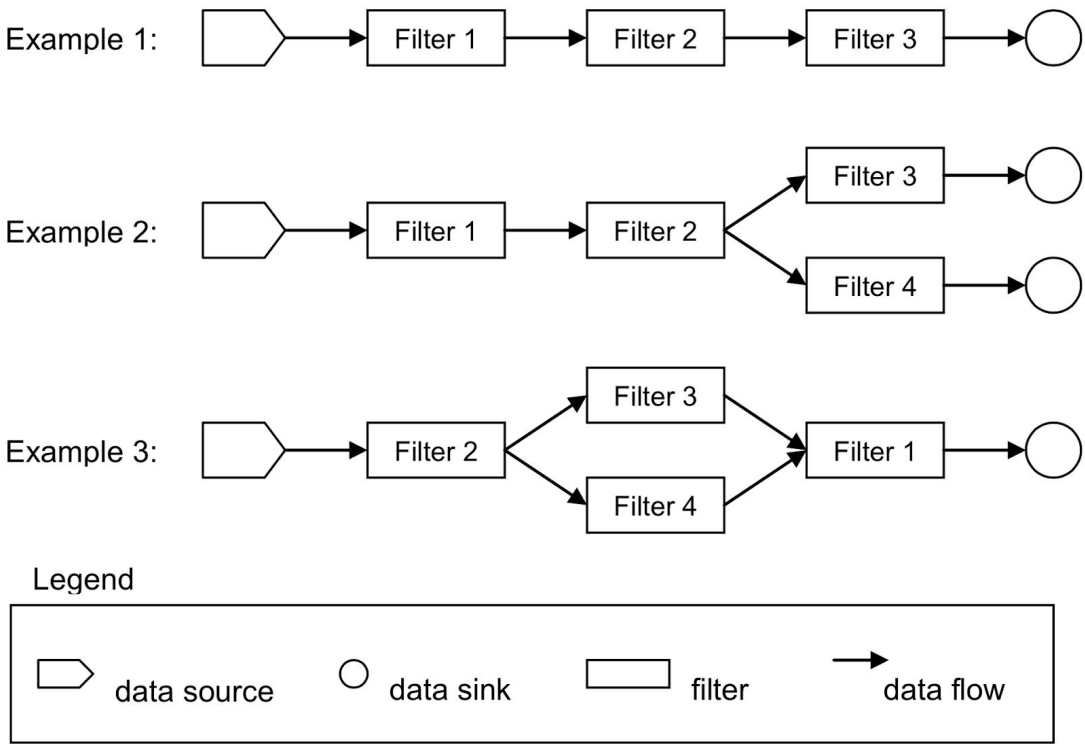
Domínio de aplicação e abordagens arquiteturais chave

O principal *stakeholder* deste projeto é uma organização que constrói sistemas de instrumentação. A Instrumentação é uma aplicação de processamento de sinal em que *streams* de dados são lidas, processadas de formas variadas, e apresentadas ou armazenadas para uso posterior. Uma parte fundamental dos sistemas de instrumentação modernos é o software que é usado para processar as *streams* de bytes de dados.

A organização pretende criar um software flexível que possa ser reconfigurado para uma variedade de aplicações e plataformas (para os nossos propósitos, podemos pensar em "plataformas" como processadores). Por exemplo, um programa pode servir para realizar a instrumentação de *streams* de dados com origem nos sensores de um automóvel (sistemas *on-board*) e que terminam na painel de instrumentos do automóvel com o *display* da temperatura, pressão do óleo, velocidade, e assim por diante. Outros filtros poderiam ser utilizados na aviação, espaço, ou aplicações marítimas. Desta forma, estes programas também poderiam apoiar o desenvolvimento e depuração de sistemas de instrumentação. Embora seja da maior importância assegurar que o sistema é altamente e facilmente reconfigurável, o sistema também terá de oferecer bom desempenhos e processar os dados o mais rapidamente possível. Para enfrentar estes desafios, o arquiteto decidiu projetar o sistema com base num estilo arquitetónico chamado

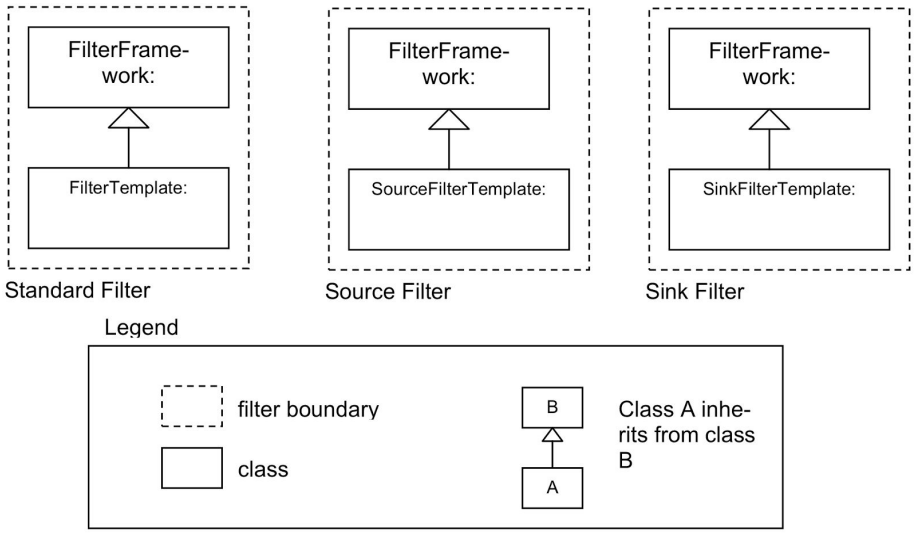
[1] Créditos: Este enunciado e os materiais fornecidos foram adaptados dos originais criados por David Garlan e Tony Latanze no âmbito do programa MSE, oferecido em parceria pela UC e a CMU entre 2008 e 2012.

pipe-and-filter. Do ponto de vista dinâmico, o sistema está estruturado da seguinte forma:



As "fontes de dados" (*data sources*) nestes sistemas são filtros especiais que lêem dados de sensores, arquivos, ou que geram dados internamente dentro do filtro. Todas as redes de filtros devem começar com uma fonte. Os "filtros" (*Filter*) que constam nestes exemplos são filtros padrão que permitem ler dados a partir de um *pipe* de *upstream*, transformar os dados, e gravar dados num *pipe* de *downstream*. Os "*data sinks*" são filtros especiais que lêem dados a partir de um filtro a montante e escrevem esses dados num arquivo ou dispositivo. Todas as redes de filtros devem terminar com um "*sink*" que consome os dados no seu formato final. Note que os fluxos podem ser divididos e fundidos, como é ilustrado na figura anterior.

O arquiteto da organização tem desenvolvido um conjunto de classes a fim de facilitar o rápido desenvolvimento de filtros e programas de forma a assegurar que estes podem ser rapidamente implementados e testados. Estas bibliotecas estão disponíveis como anexo a este enunciado. Além disso, também lhe é fornecido um exemplo que ilustra o uso destas classes. O diagrama de classes (perspectiva estática) para os filtros é o seguinte:



A classe *FilterFramework* é a classe base para todos os filtros. Esta classe contém métodos para gerir as ligações aos *pipes*, escrita e leitura de dados de e para *pipes*, e estabelecer os filtros como segmentos separados. Três filtros "modelo" foram desenhados para facilitar o trabalho de criação de filtros fonte, "*data sinks*" e filtros padrão de uma forma consistente. Cada um desses modelos de filtro descreve como produzir código para os três tipos básicos de filtros. Note-se que o modelo atual não suporta divisão ou fusão do fluxo de dados. Um novo modelo, chamado de "*PlumberTemplate*" mostra como redes de *pipe-and-filter* podem ser configuradas a partir de novos filtros. O "*Plumber*" é responsável por instanciar os filtros e conectá-los. Uma vez concluída esta tarefa, o *Plumber* termina.

Formato dos fluxos (*streams*) de dados

Os fluxos de dados do sistema seguem um formato pré-determinado composto por uma sequência de pares ID/Dado. Cada métrica tem um ID único que começa com zero. O ID de zero é sempre associado ao tempo. São-lhe fornecidos ficheiros com dados de teste que pode utilizar n execução deste trabalho. Os ficheiros estão em formato binário - uma ferramenta de visualização de dados binários é fornecida para ajudar a ler estes ficheiros e realizar tarefas de depuração. A tabela abaixo lista as métricas, IDs e tamanhos em bytes dos dados nestes ficheiros.

ID	Descrição e unidades	Tipo	Número de bytes
N/A	ID: Cada métrica tem um ID que indica para que métrica é o valor. Os IDs das métricas são listados nesta tabela, na primeira coluna à esquerda	Inteiro	4
000	Tempo: Valor em milissegundos desde a Epoch (00:00:00 GMT em 1 de Janeiro, 1970).	Inteiro long	8
001	Velocidade: velocidade do ar do veículo em nós por hora.	Double	8
002	Altitude: Distância ao solo do veículo em pés.	Double	8
003	Pressão: Pressão atmosférica à volta do veículo em PSI	Double	8
004	Temperatura: temperatura do casco do veículo em Fahrenheit.	Double	8
005	Pitch: ângulo de elevação do nariz do veículo. Pitch 0 corresponde a uma posição paralela à terra. O valor positivo	Double	8

	indica que o veículo vai a subir, negativo a descer.		

Os dados das *streams* são gravados em *"frames"* que começam com o tempo, sendo seguido por dados com IDs de entre 1 e n, com n≤5. O conjunto dados + tempo é chamado de "*frame*". O valor do tempo indica quando é que os dados do *frame* foram gravados. Este padrão é repetido até que o fim do fluxo é alcançado. Cada *frame* é escrito numa cadeia como se segue:

Frame 1	ID: 000	Time	ID: 001	Data	...	ID: n	Data
Frame 2	ID: 000	Time	ID: 001	Data	...	ID: n	Data
			:				
Frame N	ID: 000	Time	ID: 001	Data	...	ID: n	Data

Instalar o código exemplo

Em primeiro lugar, comece por extrair os ficheiros do arquivo "TP1.zip". O arquivo contém quatro diretorias: *Templates, Sample, DataSets, HexDump*. A diretoria *Templates* contém os modelos de código fonte para os filtros descritos em cima. A diretoria *DataSets* tem todos os dados de teste que vai precisar. A *HexDump* contém um utilitário que lhe permitirá ler o conteúdo dos arquivos de dados binários. A *Sample* contém um exemplo de rede de *pipe-and-filter* que ilustra a estrutura básica deste estilo. Para compilar o exemplo na diretoria *Sample*, abra uma janela da *prompt* de comando (ou inicie um terminal de linha de comando do Linux), altere a diretoria atual para *Sample* e digite o seguinte:

```
$ javac *.java
```

O processo de compilação cria os ficheiros .class. Pelo que, depois está pronto a executar o exemplo:

```
$ java Plumber > Outuput.dat
```

Este programa lê os dados do ficheiro Flightdata.dat e utiliza o estilo *pipe-and-filter* para extrair as métricas presentes no fluxo de dados existente no ficheiro. Depois apresenta no ecrã os valores para cada métrica e o seu *timestamp*.

Design e desenvolvimento

A sua tarefa é usar a estrutura existente como base para a criação de três novos sistemas. Cada novo sistema tem um ou mais requisitos. Em cada sistema, deve continuar a utilizar o estilo *pipe-and-filter* tanto quanto possível. Certifique-se de que faz uso de boas práticas de programação, incluindo comentários que descrevem o papel de cada função e de eventuais novos módulos, bem como descrevendo as alterações aos módulos do sistema base.

Sistema A

Criar uma rede de *pipe-and-filter* que leia o fluxo de dados do arquivo FlightData.dat, converte as medidas de temperatura de Fahrenheit para Celsius, e converter altitude de pés para metros. Filtrar as outras medições e escrever a saída para um arquivo de texto chamado OutputA.dat. Formato de saída é o seguinte:

Time:	Temperature (C):	Altitude (m):
YYYY:DD:HH:MM:SS	TTT.ttttt	AAAAAA.aaaaa

Sistema B

Criar uma rede de *pipe-and-filter* que faz o que o Sistema A faz, mas inclui dados de pressão. Além disso, o Sistema B deve filtrar "pontos selvagens" de pressão para fora do fluxo de dados.

Um ponto selvagem é qualquer medição de dados de pressão que exceda os 80 psi ou seja inferior a 50 psi. Para pontos selvagens encontradas na *stream*, interpolar um valor de substituição usando a média entre a última medição válida conhecida e próxima medida válida. Se um ponto selvagem ocorre no início do fluxo, substituí-lo com o primeiro valor válido; se um ponto selvagem ocorre no final da *stream*, substituí-lo pelo último valor válido. Escrever o output para um ficheiro de texto chamado OutputB.dat e formatar a saída como mostrado abaixo - marcar os valores interpolados com um asterisco, como mostrado abaixo para a segunda pressão:

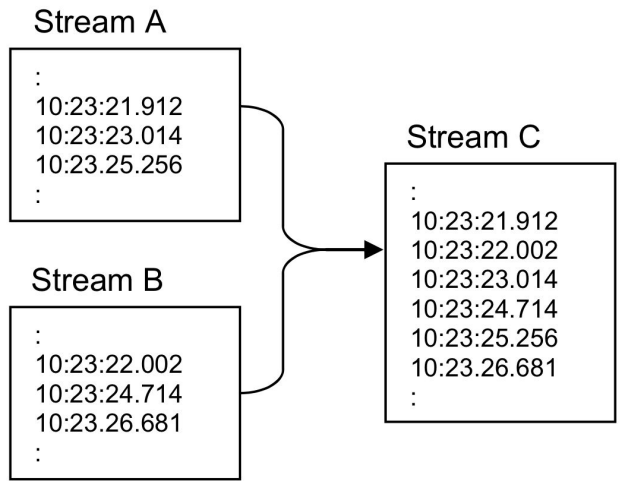
Time:	Temperature (C):	Altitude (m):	Pressure (psi):
YYYY:DD:HH:MM:SS	TTT.ttttt	AAAAAA.aaaaa	PP:ppppp
YYYY:DD:HH:MM:SS	TTT.ttttt	AAAAAA.aaaaa	PP:ppppp*
YYYY:DD:HH:MM:SS	TTT.ttttt	AAAAAA.aaaaa	PP:ppppp
:	:	:	:

Escreva também os pontos selvagens que foram rejeitados para um ficheiro texto de nome WildPoints.dat de acordo com o seguinte formato:

Time:	Pressure (psi):
YYYY:DD:HH:MM:SS	PP:ppppp

Sistema C

Criar uma rede de *pipe-and-filter* que junte dois fluxos de dados. O sistema deve ter como entrada o ficheiro SubSetA.dat e o SubSetB.dat. Ambos os ficheiros têm as mesmas 5 medições que o FlightData1.dat, que foi gravado em momentos no tempo diferente e sobrepostos. O sistema deve juntar as duas *streams* alinhar os dados temporalmente - ou seja, quando os ficheiros estiverem unidos, os dados relativos ao tempo devem aparecer de forma crescente. Isto é ilustrado em baixo com um exemplo simples. Aqui a *stream* C representa a incorporação da *stream* A e B.



Submissão dos trabalhos

O código fonte dos três programas deve ser organizado em três diretorias diferentes e comprimido num ficheiro que ZIP. Juntamente com o código fonte, deve incluir diagramas/imagens que ilustrem as soluções de design encontradas para cada programa. O arquivo resultante deverá ser submetido no Inforestudante até ao dia **23 de fevereiro** após discussão na aula PL dessa semana. Nesta discussão, os alunos deverão ser capazes de justificar as opções tomadas e identificar as alternativas consideradas nessa escolha.