MSIT-SE-M-04 Architectures for Software Systems Spring 2016

Assignment 1: Pipe-and-Filter

Discuss: Fri Feb 5, **Due:** Mon Feb 15

Problem Description

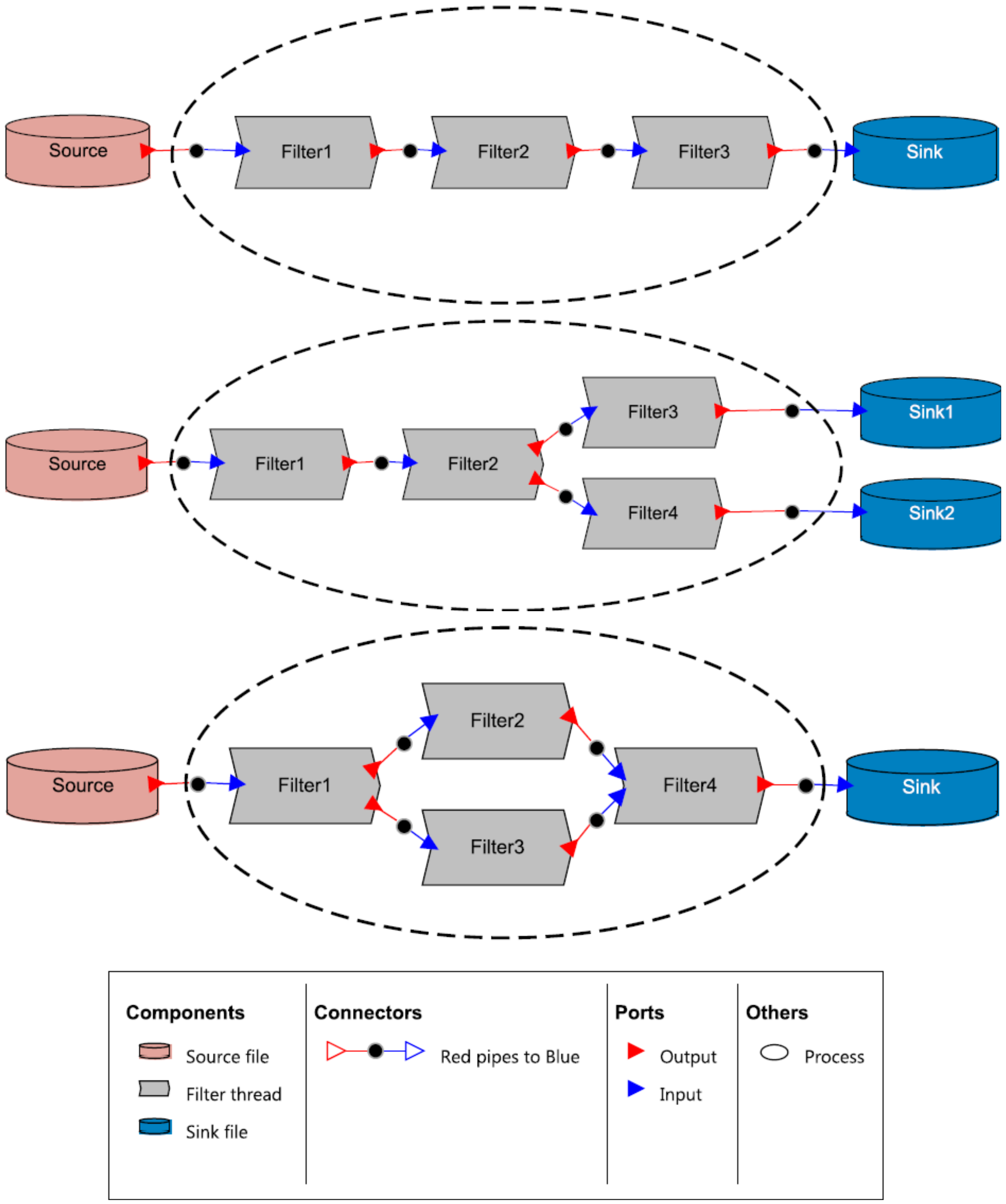
The objective of this assignment is to develop an appreciation of architectural patterns/styles, how they are used in design and analysis, and their impact on systemic properties. This assignment will utilize the pipe-and-filter architectural pattern as the overarching system structure. Part of this assignment will be implementation-oriented, allowing you to experiment with a particular pipe-and-filter implementation strategy in order to gain a clearer understanding of the issues associated with carrying an architectural design through to code. Note that this is not a programming class, and so the emphasis of this assignment is on the architecture issues – ***this cannot be stressed enough.*** Simply re-writing the entire implementation will indicate a lack of understanding of the core architectural concepts presented in class.

The assignment consists of two parts: For the first part of the assignment, you will be provided with an architectural design for a sample system together with working implementations that use a (coding) framework supporting the pipe-and-filter paradigm. The class of system for this assignment is signal processing applications, as described below. Your task in part one is to extend the existing framework to architect and build the systems specified in the requirements below. Part one will be done in teams.

The second part of the assignment consists of analyzing the architecture of the system. After your analysis, design, and coding, you will reflect upon your work and answer questions related to the design decisions your team made in part one. While you may discuss part two as a group, the write-up for part two must be done individually.

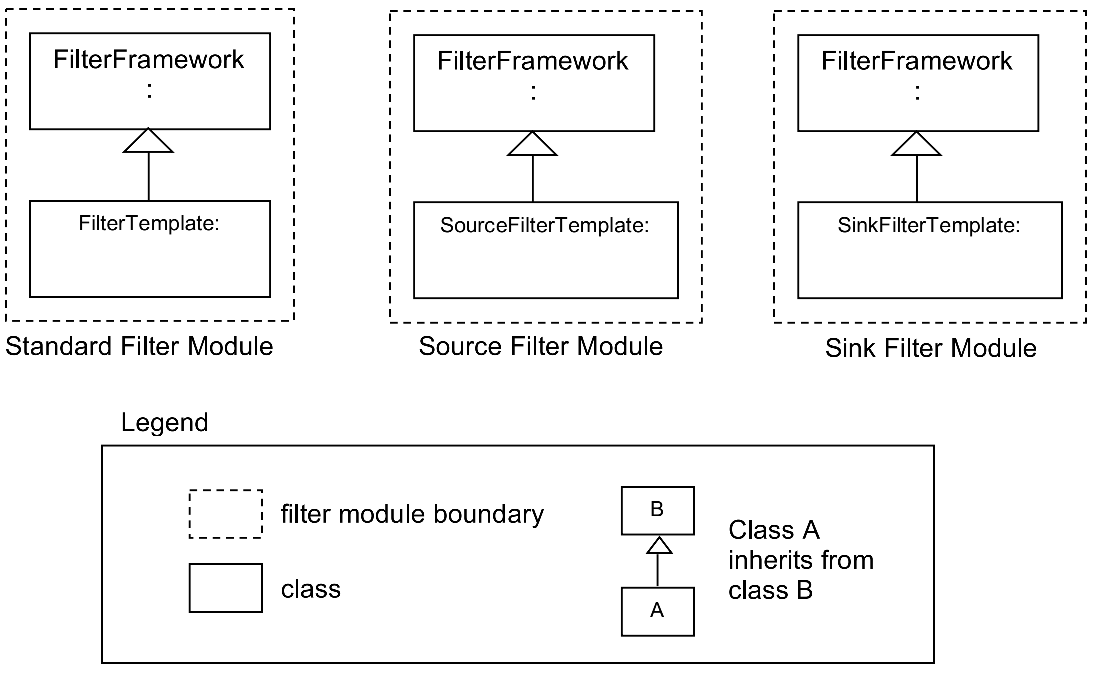
Business Context and Key Architectural Approaches

The principle stakeholder for this system is an organization that builds instrumentation systems. Instrumentation is a typical kind of signal processing application where streams of data are read, processed in a variety of ways, and displayed or stored for later use. A key part of modern instrumentation systems is the software that is used to process byte streams of data. The organization would like to create flexible software that can be reconfigured for a variety of applications and platforms (for our purposes, we can think of “platforms” as processors). For example, one application might be to support instrumentation for an automobile that would include data streams that originate with sensors and terminate in the cabin of the auto with a display of temperature, oil pressure, velocity, and so forth. Some subset of filters for this application might be used in aviation, space, or maritime applications. Another application might be in the lab reading streams of data from a file, processing the stream, and storing the data in a file. This would support the development and debugging of instrumentation systems. While it is critically important to support reconfiguration, the system must also process streams of data as quickly as possible. To meet these challenges, the architect has decided to design the system around a pipe-and-filter architectural pattern. From a dynamic perspective, systems would be structured as shown in the following examples.



The “data sources” in these systems are special filters that read data from sensors, files, or that generate data internally within the filter. All filter networks must start with a source. The “filters” shown in these examples are standard filters that read data from an upstream pipe, transform the data, and write data to a downstream pipe. The “data sinks” are special filters that read data from an upstream filter, but write data to a file or device of some kind. All filter networks must terminate with a sink that consumes the final data. Note that streams can be split and merged as shown in these examples.

The organization's architect has developed a set of modules to facilitate the rapid development of filters and applications that can be quickly tested and deployed. These libraries have been provided to you. In addition, there are several examples that have been provided to illustrate the use of these classes. The class structure (static perspective) for filters is as follows:



The FilterFramework class is the base class for all filters. It contains methods for managing the connections to pipes, writing and reading data to and from pipes, and setting up the filters as separate threads. Three filter "templates" have been established to ease the work of creating source, sink, and standard filters in a consistent way. Each of these filter templates describes how to write code for the three basic types of filters. Note that the current framework does not support splitting or merging the data stream. A fourth template, called the "PlumberTemplate" shows how pipe-and-filter *networks* can be set up from the filters created by developers. The "Plumber" is responsible for instantiating the filters and connecting them together. Once done with this, the plumber exits.

Data Stream format

The system's data streams will follow a predetermined format of measurements – pairs of measurement ID and data point. Each measurement has a unique id beginning with zero. The ID of zero is always associated with time. The table below lists the measurements, IDs, and byte sizes of the data in these files.

ID	Data Descriptions and Units	Type	Number of Bytes
N/A	Measurement ID: Each measurement has an ID which indicates the type of measurement. The Measurement IDs are listed in this table in the left column.	Integer	4
000	Time: This is the number of milliseconds since the Epoch (00:00:00 GMT on January 1, 1970).	long Integer	8
001	Velocity: This is the airspeed of the vehicle. It is measured in knots per hour.	Double	8

002	Altitude: This is the vehicle's distance from the surface of earth. It is measured in feet.	Double	8
003	Pressure: This is atmospheric pressure external to the vehicle. It is measured in PSI.	Double	8
004	Temperature: This is the temperature of the vehicle's hull. It is measure in degrees Fahrenheit.	Double	8
005	Attitude: This is the position of the nose of the vehicle relative to the surface of the earth. An attitude of 0 indicates that the vehicle is traveling level with respect to the earth. A positive value indicates that the vehicle is climbing; a negative value indicates that the vehicle is descending.	Double	8

Measurements are grouped in frames beginning with the ID, the time, and followed by data with IDs ranging from 1 to n. A set of ID, time, and data is called a frame. Time data will always appear first in a frame (ID 0) and correlates to when the data in the frame was recorded. This pattern of frames repeated until the end of stream is reached.

Note that not all the measurement files provided contain all of these measurements – you should examine the data files to see examples of what measurements may be available. The system will handle streams with arbitrary subsets of measurements (e.g. only time, velocity, and attitude). Each frame is written in a stream as follows:

Frame 1	ID: 000	Time Data	ID: 001	Data	...	ID: 005	Data
Frame 2	ID: 000	Time Data	ID: 001	Data	...	ID: 005	Data
				:			
Frame N	ID: 000	Time Data	ID: 001	Data	...	ID: 005	Data

Test measurement files have been provided. These contain test flight data that you will use for the project. The file data is in binary format – a data dump tool is provided to help read these files. It's located in located in the directory "HexDump."

Installing the Source Code

First, unzip the "A1 Source.zip" file into a working directory. You will see five directories: Templates, Sample1, DataSets, HexDump. The *Templates* directory contains the source code templates for the filters described above. The *DataSets* directory has all of the test data that you will need. The directory *HexDump* contains a utility that will allow you to read through the contents of the binary data files. The directory *Sample1* contains a working pipe-and-filter network example that illustrates the basic framework. To compile the example in the Sample1 directory, open a command prompt window (or start a Linux command line terminal), change the working directory to Sample1, and type the following:

```
...\assignment1\sample1> javac *.java
```

The compile process above creates the class files. After you compile the system, you can execute it by typing the following:

```
...\assignment1\sample1> java Plumber
```

Sample1 is a basic pipe-and-filter network that shows how to instantiate and connect filters, how to read data from the Flightdata.dat file, and how to extract measurements from the data stream.

The output of this example is measurement data and time stamps (when each measurand was recorded) – all of this is written to the terminal. If you would like to capture this information to assist you in debugging the systems, you can redirect it to a file as follows:

```
...\assignment1\source> java Plumber > output.txt
```

In this example, the output is redirected to the file output.txt.

Part 1: Design and Construction (TEAM TASK)

The task of your team is to use the existing framework as the basis for creating three new system variants. Each new system has one or more requirements. In each system, please adhere to the provided framework (pipe-and-filter architectural pattern) as closely as possible. Make sure that you use good programming practices including comments that describe the role and function of any new modules, as well as describing how you changed the base system modules. If you do not follow reasonably good programming practices, your team will be penalized.

System A

Create a pipe-and-filter network that will read the data stream in FlightData.dat file, convert the temperature measurements from Fahrenheit to Celsius, and convert altitude from feet to meters. Filter out all other measurements (no need to save these). Write the output to a text file called OutputA.dat. Format the output as follows:

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

System B

Create a pipe-and-filter network that does what System A does but includes pressure data in the output. In addition, System B should filter “wild points” out of the data stream for pressure measurements. A wild point is any pressure data that varies more than 10PSI between samples and/or is negative. For wild points encountered in the stream, extrapolate a replacement value by using the last known valid measurement and the next valid measurement in the stream. Extrapolate the replacement value by computing the average of the last valid measurement and the next valid measurement in the stream. If a wild point occurs at the beginning of the stream, replace it with the first valid value; if a wild point occurs at the end of the stream, replace it with the last valid value. Write the output to a text file called OutputB.dat and format the output as shown below – denote any extrapolated values with an asterisk by the value as shown below for the second pressure measurement:

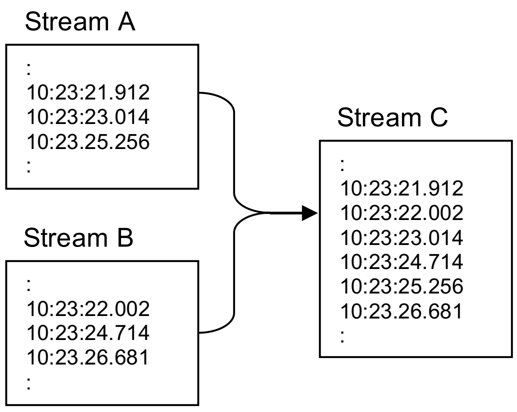
Time:	Temperature (C):	Altitude (m):	Pressure (psi):
YYYY:DD:HH:MM:SS	TTT.ttttt	AAAAAA.aaaaa	PP:ppppp
YYYY:DD:HH:MM:SS	TTT.ttttt	AAAAAA.aaaaa	PP:ppppp*
YYYY:DD:HH:MM:SS	TTT.ttttt	AAAAAA.aaaaa	PP:ppppp
:	:	:	:

Write any rejected wild point values and the time that they occurred to a second text file called WildPoints.dat using a similar format as follows:

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

System C

Create a pipe-and-filter network that merges two data streams. The system should take as input the SubSetA.dat file and the SubSetB.dat. Both of these files have all 5 measurements listed above, which was recorded at different, but overlapping times. The system should merge these two streams together and time-align the data – that is, when the files are merged the single output stream's time data should be monotonically increasing. This is illustrated below with a simple example. Here Stream C represents the merger of Stream A and Stream B.



In addition to merging the streams, you must apply filters to the data as follows:

Filter out ALL data collected below 10,000 feet. Write all these measurements to a file named "LessThan10K.dat."

Filter pressure measurement wildpoints and replace any filtered pressure wildpoints with extrapolated values as you did in System B. Write the filtered pressure values to a file named "PressureWildPoints.dat."

Packaging and Submitting Part 1

- Systems A, B, and C should be clearly separated, both in terms of implementation and write-up (as described above). Place each implementation in a different folder.

- Part 1 must be posted to Moodle in a zip file (only zip, not rar or any other archive format) following the naming convention: TEAM-NAME_ASSIGNMENT-NUMBER. For example, for team *FooFighters,* the archive name would be *FooFighters_A1*.

- We will test your programs on our computers with test data sets that are the same or similar to the data sets you have been provided with. **You must clearly describe how to run your program. PLEASE DO NOT ASSUME THAT WE HAVE ANY PARTICULAR RUNTIME ENVIRONMENTS OR APPLICATIONS INSTALLED ON OUR SYSTEMS UNLESS SPECIFICALLY CALLED FOR IN THE ASSIGNMENT. WE WILL NOT INSTALL YOUR RUNTIME ENVIRONMENT ON OUR SYSTEM TO RUN YOUR PROGRAMS.**
  We will run your programs from the **command window** for this assignment – so please keep this in mind. In general, **we will not recompile your code** on our machines. If your system does not run, or we cannot figure out how to run your programs, you will be penalized. If you have any questions or issues, please ask.

Part 2: System Analysis (INDIVIDUAL TASK)

Cooperation Notes: Even though the systems were developed by the team, you are free to agree or disagree with your team's implementation decisions in your individual write-up. You may use the team diagrams in your individual write-up. Feel free to alter the team diagrams in your individual write-up to support your analysis. You should not share any diagrams you create individually with your teammates. <u>Please include your team name in your individual write-up to facilitate its cross-referencing with the team tasks</u>:

Please answer the following questions. Each question has several parts. Make sure that you answer each question completely in your write-up:

1. For the overall system discuss the following:
   - According to the business context, what are the key architectural drivers of the system and what is their relative priority?
   - How well does the architect's initial design support the business goals as described in the business context?
2. For each system A, B, and C:
   - Describe the architecture of the system after your modifications. Be sure to include appropriate views of the system to support your analysis and discussion of the design. You are free to use whatever notations you prefer, but you must follow the best practices of architectural design documentation as presented in class.
   - Discuss how well the design choices that you made support the business goals of the system in terms of the relevant quality attributes. What were the key design decisions, tradeoffs, and what motivated your choices?
3. Given your design and implementation of pipe-and-filter systems used for this assignment:
   - To what extent do your implementations (A, B, and C) differ from what is implied by an idealized notion of pipes and filters? Explain why and the impact it might have on systemic properties of your systems.
   - Are there any other possible solutions that you could have adopted? What made you decide on your solutions over these other possible solutions?
4. Extension 1 (you are not required to implement this extension; however, describe the system in sufficient detail so that the architecture can be understood):
   - Suppose that measurement data varied from data file to data file so we needed to supply measurement metadata describing measurement ID, byte size, type, units, and so forth to each filter. How might you pass along measurement metadata to each filter? Sketch the architecture of this system, describe, and justify your design decisions and analysis.
   - How would this requirement impact the key quality attributes of the system? How would the existing framework be affected? What mechanisms of the existing system might be affected by this requirement?
   - Would this requirement force you to deviate from the idealized notion of a pipe-and-filter pattern? Explain why or why not and justify your answer.
5. Extension 2 (you are not required to implement this extension; however, describe the system in sufficient detail so that the architecture can be understood):
   - Suppose there is a new requirement to run filters on physically separate processors as well as supporting co-located filter networks. How might you satisfy this requirement? Sketch the architecture of this system, describe, and justify your design decisions and analysis.
   - How would this impact the important quality attributes of the system? How would the existing framework be affected? What mechanisms of the existing system might be affected by this requirement?
   - Would this requirement force you to deviate from the idealized notion of a pipe-and-filter pattern? Explain why or why not.

Packaging and Submitting Part 2

- Part 2 must be posted to Moodle as a **Microsoft Word document. No other word processor format will be acceptable.**

- Post your file using the following naming convention: LASTNAME_ASSIGNMENT-NUMBER. As an example Harry Bovick A1 assignment write-up name would be *Bovic_A1*.

Grading Criteria

Your assignment will be graded based upon:

- The quality and contents of your write-up… this includes:
    - the quality of the design views and descriptions
    - discussions of the trade-offs you/your team made and implications of changes to the system's architecture and the effect on systemic properties
    - discussion of the deviations from the pipe-and-filter architectural pattern if and where this happens as well as their effects on systemic properties
    - clarity, conciseness, and completeness of the write-up including the proper use of English (the absence of grammatical issues and misspellings)
- The consistency between the design representations and the implementation.
- The degree to which your solutions adhere to the pipe-and-filter architectural pattern where possible to do so.
- The correct operation of the solutions - we will test your solutions with our test data.
- Professionalism, which includes the quality of the team presentation, timely submission of team and individual assignments, and well-structured and documented source code.

Each assignment will be weighted as follows (100 points maximum):

Part 1: Team Implementation

- Implementation of System A: 10 points
- Implementation of System B: 15 points
- Implementation of System C: 25 points

Part 2: Individual Write-up

- Question 1: 10 points
- Question 2: 10 points
- Question 3: 10 points
- Question 4: 10 points
- Question 5: 10 points