EE249 Homework 1

Alessandro Pinto
{apinto@eecs.berkeley.edu}

September 10, 2004

Goal of HW1 is to give you some examples of orthogonalization of concerns and platform-based design. **General Comments:** *an answer to a problem is neither right nor wrong (but there are exceptions of course). I would rather consider your homework as solutions to design problems. I could argue that your solution is inefficient and you could argue the same about mine. Grade is established mostly on the reasoning that you follow to answer the questions. Hence it is in your interest to justify your claims. You can use any kind of sources as long as you include references.*

1. **Computation vs. Communication.** Consider a simple producer-consumer example. Producer and consumer are abstraction of computational blocks that might be very complex: think of two mobile phones for instance. In our case the producer is very simple: it generates random data and we assume the existence of a library function `randomdata()` which returns a random object (where the returned type could be whatever you need it to be). Producer generates data on demand. The consumer is just a sink. It receives data and has a way of signaling that the reception is completed.

   Assume also that there is a way of checking whether the received data is correct or not, namely there exists a library function `checkdata(data)` which returns `true` if the data is correct and `false` if it is corrupted.

   Consider two protocols:

   - Simple dropping: consumer checks if the received data is corrupted in which case the data is dropped and a completion signal is sent back to the producer.
   - Retransmission: consumer checks if the received data is corrupted in which case it asks the producer to send the data again until the reception is successful and a completion signal is sent back to the producer.

QUESTION 1.1: write the two concurrent processes `producer` and `consumer` in the case of simple dropping protocol. The producer will have an output signal for sending data, an input signal for receiving the acknowledgment and an input signal for receiving the completion (end of transmission). The receiver will have of course the specular ports to match the producer. Use pseudocode (as long as it is understandable) to describe the two processes.

QUESTION 1.2: write the two concurrent processes `producer` and `consumer` in the case of retransmission protocol.

As you may have noticed, in order to answer question 1.2 you had to rewrite both processes almost from scratch. This is not a big deal in a simple case like producer-consumer but it breaks reusability. In this case all the producer does is to generate random data. In real systems it might represent a more complex computation and you may want to be able to reuse that component in other projects without touching it. We didn't separate communication from computation. Producer functionality is to produce data on demand while consumer functionality is to receive data and basically ask for another one:

QUESTION 1.3: write a process `producer` that has an input named `gendata` and an output named `data`. Producer has to generate a new random data whenever the signal `gendata` is enabled.

QUESTION 1.4: write a process `consumer` that has an input named `data` and an output named `completed`. Consumer has to enable the `completed` signal each time that a data is received.

Note that those two processes can be connected together. The composed system is an autonomous system where producer keeps on producing data and receiver keeps on receiving data. We just modeled the functionality of both processes and connected them together. Now think of the protocol between the two components as two new blocks like in figure 1.

QUESTION 1.5: write the up and down converter in the case of simple dropping and retransmission. You can define the protocol handshaking as you like.

2. **Function vs. Architecture.** We want to represent a graph. We then the following building blocks:

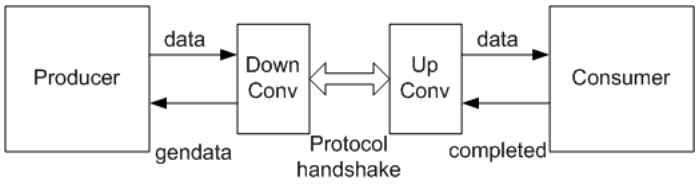

Figure 1: Two other processes are introduced: a down converter and an up converter.

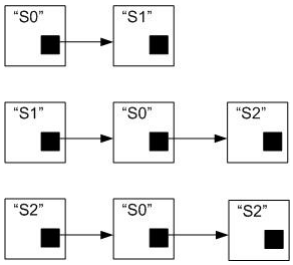

Figure 2: Adjacency list representation of the graph.

- A data structure containing a label to name the node and a pointer to a list of adjacent nodes.

This is also called the adjacency list representation of a graph and is used in many computer programs to store a graph in an internal data structure. Figure 2 shows an example of a directed graph implemented using this method.

QUESTION 2.1: Figure 2 is an implementation of a specification using a particular architecture. What is the specification? (Hint: recall the definition of a graph).

The following algorithm is the classical breath first search (BFS). It is implemented using the adjacency list representation of a graph. Also it assumes that there is a queue data structure to store nodes that are visited.

QUESTION 2.2: This is again an implementation of the BSF. What is it's purely functional description?.

QUESTION 2.3: Based on the answer to question 2.2 can you identify a set of library elements that an architecture should have in order to implement BFS? Using this components, can you implement other algorithms?

Algorithm 1 BFS

```
1: BFS(G,s)
2: for all u ∈ V(G) do
3:    color[u] ← WHITE
4:    π[u] ← NIL
5: end for
6: color[s] ← GRAY
7: π[s] ← NIL
8: Q ← {s}
9: while Q ≠ ∅ do
10:   u ← head[Q]
11:   for all v ∈ Adj[u] do
12:      if color[v] = WHITE then
13:         color[v] ← GRAY
14:         π[v] ← u
15:         ENQUEUE(Q, v)
16:      end if
17:   end for
18:   DEQUEUE(Q)
19:   color[u] ← BLACK
20: end while
```

3. **Platform Based Design.** We want to apply the PBD concept to an example taken from civil engineering (note that this is an example and it is not the way they follow to build houses).

   Figure 3 shows our setting. The function is a *build a 300 square meter of living space*. The platform has the following library:

   - Walls: they are characterized by height and a width
   - Floors: they are characterized by width and length
   - Roofs: they are characterized by width, length and height.

   We have the following composition constraints:

   (a) floors have to lie on walls and must be horizontally placed.

   (b) Walls have to lie on floors and must be vertically placed.

   (c) Every platform instance has start with a floor and terminate with a roof.

   The **cost** of the platform instance is the total surface of its components. Note that surface is an additive quantity like energy.

   We have the following constraints to satisfy:

   (a) Given any two points of the living space, their distance should not be greater than 10 meters.

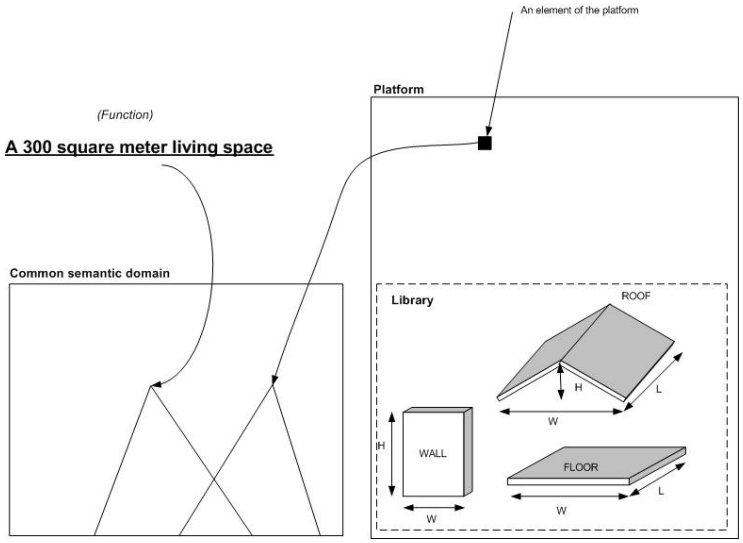

Figure 3: Function and platform

Our objective is to find a minimum cost solution that satisfies the constraints. The common semantic domain is composed of all living spaces obtained as justaposition of parallelepiped (either horizontal or vertical).

QUESTION 3.1: Given the functional description , describe the set that is gotten by mapping it into the common semantic domain.

QUESTION 3.2: Characterize at least two different platform instances.

QUESTION 3.3: Given a platform instance, characterize the set gotten by mapping it into the common semantic domain.

QUESTION 3.4: Is Evans Hall an instance of this platform? And what about Soda Hall?

QUESTION 3.5: Pick an implementation of the function in the common semantic domain that satisfies the constraints.

QUESTION 3.6: Compute the cost of your implementation.