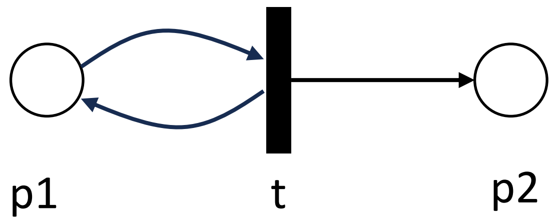
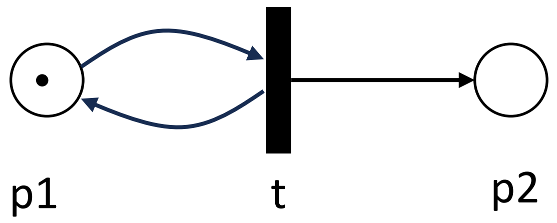
Exercises on architectural verification

Petri nets

- Define a Petri net and the corresponding initial marking so that
  - It is bounded: there are no places that can accumulate an unbounded number of tokens
- Is it possible to define an initial marking that make it unbounded? If not, modify the Petri net to make it unbounded
- Define a Petri net that is deadlock free (live) and modify it to introduce a deadlock

Petri nets - solutions



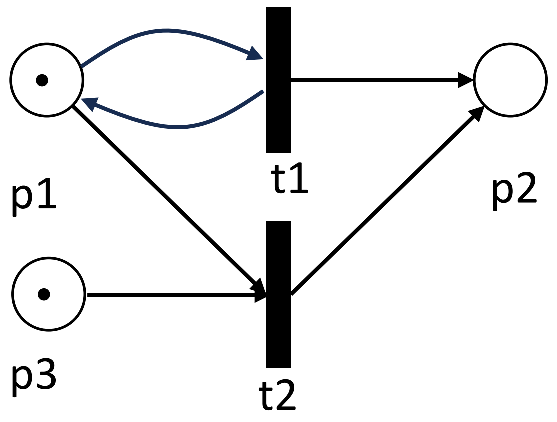


This is also deadlock free

Empty marking (no tokens), no transition can fire, then bounded

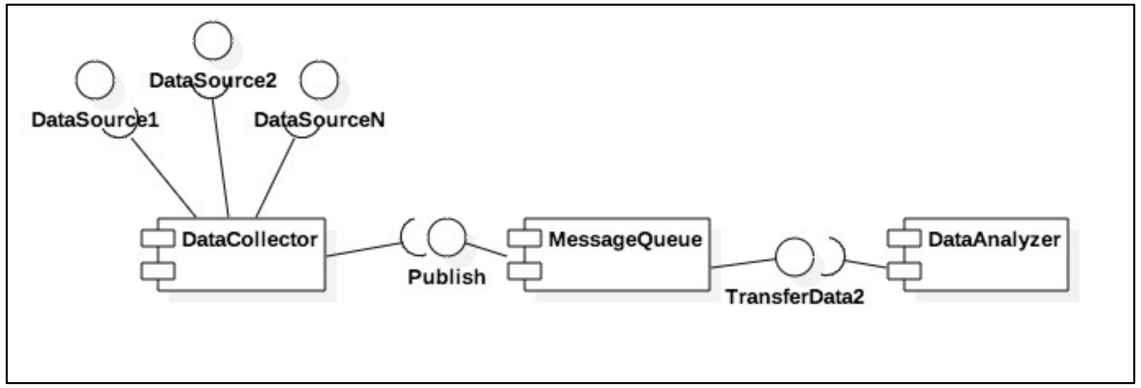
One token in p1, t can fire at any step, tokens are accumulated in p2: unbounded

Introducing a deadlock…



When t2 fires, no further transition can fire, the net is dead

Availability (from the SE2 exam of June 27th, 2018)



- Consider the system above. Assume that the participating components offer the following availability:
  - DataCollector: 99%
  - MessageQueue: 99.99%
  - DataAnalyzer: 99.5%
- Provide an estimation of the total availability of your system (you can provide a raw estimation of the availability without computing it completely).
- Assuming that you wanted to improve this total availability by exploiting replication, which component(s) would you replicate? Please provide an argument for your answer.
- How would such replication impact on the way the system works and is designed?

Availability - solution

- Data must flow through the whole chain of components to be processed. This implies that we can model the system as a series of component.
- The total availability of the system is determined by the weakest element, that is, the DataCollector.
  - A_{Total} = 0.99*0.9999*0.995 = 0.985
- If we parallelize the data collector adding a new replica, we can achieve the following availability:
  - A_{Total} = (1-(1-0.99)2) *0.9999*0.995 = 0.995

Availability - solution

- Impact of parallelization:
  - If both DataCollector replicas acquire information from the same sources (hot spare approach), then the other components will see all data duplicated and will have to be developed considering this situation.
  - For instance, the MessageQueue could discard all duplicates.
  - Another aspect to be considered is that both DataSources and MessageQueue have to implement mutual exclusion mechanisms that ensure the communication between them and the two DataCollector replicas does not raise concurrency issues.
  - Another option could be that only one DataCollector replica at a time is available and the other is activated only when needed, for instance, if the first one does not send feedback within a certain timeout (warm or cold spare).