Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

Prof. Elisabetta Di Nitto, Matteo Rossi and Damian Tamburri

20133 Milano (Italia)
Piazza Leonardo da Vinci, 32
Tel. (39) 02-2399.3400
Fax (39) 02-2399.3411

Software Engineering 2 – Written Exam 1 (WE1)

January 24th, 2022

Notes

1. Remember to write your name and Id number (matricola) on each piece of paper that you hand in.
2. You may use a pencil.
3. Incomprehensible handwriting is equivalent to not providing an answer.
4. The use of any electronic apparatus (computer, cell phone, camera, etc.) is strictly forbidden, with the exception of an ebook reader.
5. The exam is composed of three exercises. Read carefully all points in the text!
6. **Total available time for WE1: 1h and 30 mins**

Question 1 Alloy (8 points)

A software company aims to create an application that manages the configuration of system deployments. A system deployment is made of components, where each component executes one or more functions.
A function, to be executed, might require the presence in the deployed system of other functions (for example, an "alarm notification" function requires that an "alarm generation" function is also deployed in the system, executed on some component). Given a function *f*, we call the set of functions that are required by *f* the "technological requirements" of *f*.
The specification of the "minimum configuration" of a system deployment consists of: (i) a set *F* of functions that must appear in the desired system deployment; and (ii) for each function *f* in *F*, the minimum number of instances of *f* that must appear in the deployment – i.e., the minimum number of components executing *f* that must appear in the deployment.

Alloy_1 (4 points)
Define suitable Alloy signatures – and any related constraints – to describe functions, components, technological requirements, system deployments, and specifications of minimum configurations.

Alloy_2 (2 points)
Define an Alloy predicate *isSpecificationMet* that, given a system deployment and the specification of a minimum configuration, returns true if the deployment meets the specification.

Alloy_3 (2 points)
Define an Alloy predicate *isConfigurationValid* that, given a system deployment, returns true if all technological requirements of the functions that appear in the system are met.

Solution

Alloy_1

```
sig Function {
  reqs : set Function
} { not this in reqs }

sig Component {
   execs: some Function,
}

sig MinConfigSpec {
  funcs: Function -> one Int
} { all f : funcs.Int | funcs[f] > 0 }

sig Deployment {
  comps: some Component
}
```

Alloy_2

```
pred isSpecificationMet[ d: Deployment, spec: MinConfigSpec ] {
  all f: spec.funcs.Int |
      spec.funcs[f] <= #(d.comps & execs.f)
}
```

Alternatively, the predicate could be defined in this (and also others) way:

```
pred isSpecificationMet[ d: Deployment, spec: MinConfigSpec ] {
  all f: spec.funcs.Int |
      let cs = { c: d.comps | f in c.execs } |
      spec.funcs[f] <= #cs
}
```

Alloy_3

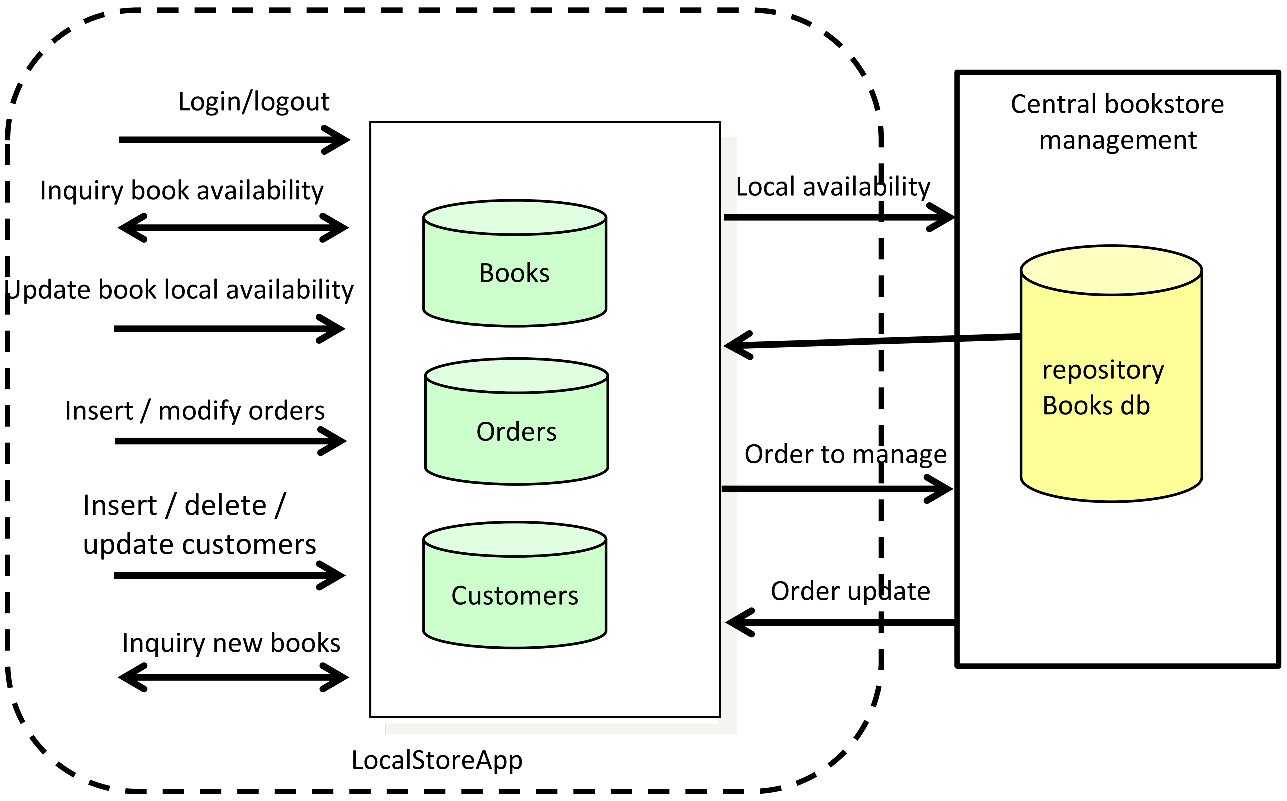
```
pred isConfigurationValid[ d: Deployment ] {
  all f: d.comps.execs | f.reqs in d.comps.execs
}
```

Question 2 Function Points (5 points)
We want to estimate the size of a bookstore application in terms of Function Points. We focus on a chain of bookstores. The application under analysis is called **LocalStoreApp** and, as the name suggests, can be used by the local bookstores of the chain.
Another software, called *CentralBookstoreManagement*, is running in the central hub.

LocalStoreApp manages directly a local database composed of the following elements: books, including the typical information relevant to a book and its local availability; customers, including their typical contact information; orders, including order id, order status, book id, customer id, communication type.
The application allows the bookstore manager to login/logout, perform book availability inquiries (in particular, it allows the bookstore manager to visualize the availability of books in the local store, as well as the one in the central store through CentralBookstoreManagement), input / change orders that will be sent to CentralBookstoreManagement, input/change/delete customers, change the information about the local availability of books, visualize the new available books.
CentralBookstoreManagement keeps updated the information about books, as well as their availability in the central storehouse. Moreover, it handles the orders coming from the local bookstores and receives updates about the availability of books in local stores.
As mentioned above, LocalStoreApp receives from CentralBookstoreManagement lists of book ids, with information about their availability in the central storehouse and their price.
In case the local book quantity is zero, the bookstore manager can use LocalStoreApp to place an order, which is stored locally and sent to CentralBookstoreManagement. CentralBookstoreManagement, in turn, handles the order, creates an order to be sent out – of which the central bookstore clerks will take care – and sends an order update to LocalStoreApp.
When the book arrives to the local store, the bookstore manager updates the local book availability and the order status.

Draw a diagram of the LocalStoreApp in terms of function points and identify the type and complexity of each function point, providing a short explanation.

Solution



1. 3 Internal Logical Files: Books, Orders, Customers. We can argue that the structure of Customers is simple. Books can be considered simple; however, they could also be considered of medium complexity as the number of entities is potentially high. Based on this consideration, Orders can be considered as medium as well. So, we have 1 simple and 2 medium: 7+2*10 = 27 FPs (but 7*3 = 21 FPs if all data are considered as simple).
2. 1 External Interface File: The Books db made available by the Central bookstore management. This has a simple structure 1x5 = 5 FPs.
3. 2 External Outputs: The LocalStoreApp sends two outputs to the Central bookstore management: order to manage, local availability: one source file for each of them. So, we can consider them as simple complexity: 2x5 = 10FPs.
4. 9 External Input: login/logout, insert / modify orders, insert / update / delete customers, order update (this input comes from the Central bookstore management), update local availability. These operations are generally quite standard so we assume they are simple: 9x3 = 27 FPs.
5. 2 External Inquiries: inquiry book availability and inquiry new books. Also these operations are simple queries, so we assign a simple complexity: 2x3 = 6 FPs.
6. Total: 75 FPs (69 if we consider Books and Orders as simple).

Question 3 JEE (3 points)

Assuming that we are implementing a client-server e-commerce application in JEE, can we use a stateless bean to implement the cart logic? If yes, how can we implement it? Is the solution efficient (provide a short explanation)?

Solution

Yes, it is possible to use a stateless bean if we persist the state of the conversation with the client on some database. This way, when the client proceeds with the order of the products in the cart, the server will be able to find the needed data in the database.

The main problem of this solution is efficiency. Any instance of the stateless bean will have to connect to the database. These connections may become a bottleneck for the application. Moreover, most of the data concerning the cart status may not be needed in the future. Therefore, persisting them may be overkill.