University of Reading

Course: BSc. Computer Science
Module: Concurrent Systems
Module Convenor: Dr. Muniyappa Manjunathaiah
Student: Vinícius de Oliveira Silva – **Student Number:** 24023627

Car Park Coursework Report

1. Introduction:

In the present day, computers with multiple processing cores are a reality in the industry. Simply increasing the processors' clock is not enough to supply the market's needs and it was found out that parallel processing is an efficient solution that provides expressive performance gains with a reasonable cost.

The problem with this approach is that although multi-core processors are capable to provide unprecedented performance, they also require extra effort when developing applications that are capable of taking advantage of their extended processing power. Developing software solutions that make efficient use of multiple core processors is not an easy task. The programmers must be able to identify parallelization opportunities within the problem they are working on and also make sure that the concurrent running threads do not introduce a non-deterministic behaviour to the application.

Because of this sort of problems, the industry created some resistance to concurrent applications, believing that often the benefits of having a faster multi-threaded application are not worth the effort required to develop it. Fortunately, instead of completely abandoning concurrent systems, computer scientists developed new theories and techniques that minimize the problems caused by multi-threading, making it much easier for developers to write applications that use the multiple cores provided by modern processors.

One of the new technologies that emerged from the effort of those scientists is CSP – Communicating Sequential Processes. This technology works by assuming that each thread is a simple sequential process that is able to communicate with other threads. This approach eliminates the use of shared variables, making writing multi-threaded systems a much simpler task.

In this coursework, the CSP technology will be used to produce a concurrent web application that simulates a virtual car park system. The main goal of this project is to show how the use of this technology helps the development of concurrent systems.

2. Modelling:

Accordingly to the coursework specification, the system should be modelled as three parallel components: Online Booking, E-Ticket and Car Park. Each component is a sequential process (or a composition of sequential processes) able to communicate with each-other by using alphabetized communication.
In order to implement this kind of communication, it is necessary to assign an alphabet to each of the components. An alphabet is the set of events that a process uses. In an alphabetized communication, when two processes which have some events in common need to communicate, they must synchronize so that no erroneous behaviour will be presented by the system.
The components used to create the system are described below.

- **Online Booking:**
  This component operates a service on the computer's port 8080 that provides the user the ability to book a place in the car park. The user must enter some identification data and the hours for the booking. Using alphabetized communication, this component sends the booking request to the Car Park component and waits for its reply informing whether or not the booking was accepted.
  The events that compose this process' alphabet are:
  αOnline Booking = {Booking request, Processed booking}

- **Car Park Composition**
  This component is actually a parallel composition of the three sub-processes detailed below:
  - **Arrival:**
    This process operates a service on the computer's port 8082 that provides the user the ability to check-in at the car park. The user must enter the booking number generated by the booking component, the driver's licence number and the number of the booked parking space.
    Using alphabetized communication, this component sends the check-in request to the controller process and waits for its reply informing whether or not the check-in was accepted.
    The events that compose this process' alphabet are:
    αArrival = {Arrival request, Processed arrival}

  - **Departure:**
    This process operates a service on the computer's port 8083 that provides the user the ability to check-out at the car park. The user must enter the booking number generated by the booking component, the driver's licence number and the number of the booked parking space.

Using alphabetized communication, this component sends the check-out request to the controller process and waits for its reply informing whether or not the check-out was accepted.
The events that compose this process' alphabet are:
αDeparture = {Departure request, Processed departure}

- **Controller:**
  This process can be considered the core of the system as it is responsible for taking decisions regarding bookings, arrivals and departures. This process is also responsible for the interaction between the Car Park composition and the other system components which makes a good alphabetized communication an essential matter for this process.
  The Controller component uses auxiliary data structures in order to model the system in an object-oriented way.
  This component's alphabet is:
  αController = {Arrival request, Processed arrival, Departure request, Processed departure, Booking request, Processed booking, ETicket request}.

- **ETicket Composition:**
  This component is also a parallel composition of three sub-processes. More details of each one are provided below:
  - **ETicket:**
    This process is responsible for issuing an eTicket to the customers that book a place at the car park. It communicates with the Car Park Composition component, waiting for it to send a request, and whet it does, this process communicates to the MailBox component, simulating the issue of a message to the customer.
    As an extra feature, the ETicket process also uses an additional Java library to send an actual email to the customer containing a simplified representation of the eTicket.
    The events that compose this process' alphabet are:
    αETicket = {ETicket request, ETicket sent}

  - **MailInt:**
    The process MailInt (Mail Interface) provides interaction between the user and the ETicket Composition process. It operates a service on the computer's port 8081 that allows the user to see the eTicket generated by the ETicket sub-process.
    The events that compose this process' alphabet are:
    αMailInt = {ETicket retrieval request, ETicket retrieved}

- **MailBox:**

  This process is used to store in memory the eTickets for all the customers. It communicates to the two other sub-processes that make up the ETicket composition in order to provide message transferring functionalities to the system.

  The events that compose this process' alphabet are:

  αMailBox = {ETicket sent, ETicket retrieved}.

2.1 CSP Modelling:

Now that the alphabets of each component are defined, it is possible to design a model of the system by using the CSP notation.

ONLINE BOOKING=

Booking request → Processed booking → ONLINE BOOKING;

CAR PARK COMPOSITION = ARRIVAL $_{\alpha Arrival}$ || $_{\alpha Controller}$ CONTROLLER $_{\alpha Controller}$ || $_{\alpha Departure}$ DEPARTURE;

ARRIVAL =

Arrival request → Processed arrival → ARRIVAL;

CONTROLLER =

 Arrival request → Processed arrival → CONTROLLER;
|Departure request → Processed departure → CONTROLLER;
|Booking request → Processed booking → CONTROLLER;
|ETicket request → CONTROLLER;

DEPARTURE =

Departure request → Processed departure→ DEPARTURE;

ETICKET COMPOSITION = ETICKET $_{\alpha ETicket}$ || $_{\alpha MailBox}$ MAILBOX $_{\alpha MailBox}$ || $_{\alpha MailInt}$ MailInt;

ETICKET =

ETicket request → ETicket sent → ETICKET;

MAILBOX =

ETicket sent → ETicket retrieved → MAILBOX;

MAILINT =

ETicket retrieval request → ETicket retrieved → MAILINT;

SYSTEM =

ONLINE BOOKING $_{\alpha Online\ Booking}$ || $_{\alpha Car\ Park\ Composition}$ CAR PARK COMPOSITION $_{\alpha Car\ Park\ Composition}$ || $_{\alpha ETicket\ Composition}$ ETICKET COMPOSITION

3. Implementation and Testing

3.1 Environment:

The environment used to implement the presented model consists in an Intel Core i7 based computer running Java 8 under Windows 10 pro x64.

3.2 Implementation Details:

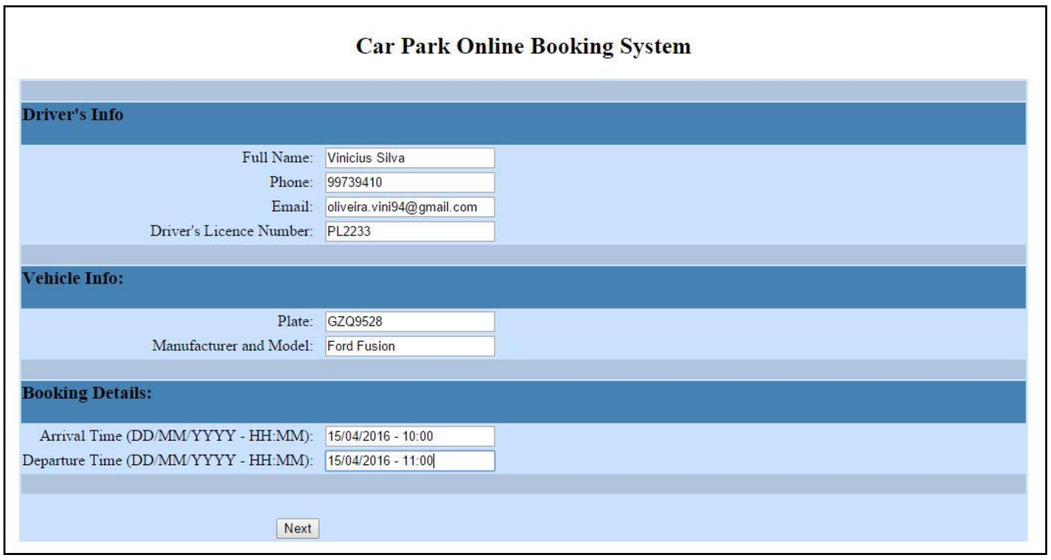
The designed model was physically implemented by using Eclipse Mars along with the JCSP library version 1.1 and the JavaMail API version 1.5.5.

3.3 Testing:

In order to test the system, it is necessary to double-click the file ViniciusSilva.jar and open a web browser (the system is certified to run on Google Chrome, and might not work well on other browsers) and type "http://localhost:8080" in the address bar.
The booking screen should be displayed and by filling the forms appropriately, it is possible to verify the behaviour of the system.
The screenshots presented below show how the system behaves and its capability to meet the required features.

3.3.1 Booking:

After typing "http://localhost:8080" on the browser address bar, the booking screen should be displayed. After filling the form properly, the user should click on the "Next" button placed on the bottom of the page. The picture below shows an example of how it works.



Car Park Online Booking System

Driver's Info

Full Name:	Vinicius Silva
Phone:	99739410
Email:	oliveira.vini94@gmail.com
Driver's Licence Number:	PL2233

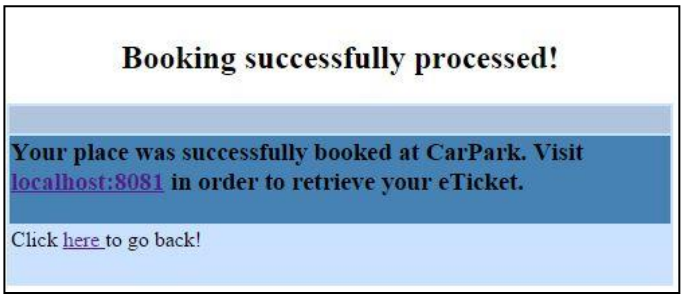
Vehicle Info:

Plate:	GZQ9528
Manufacturer and Model:	Ford Fusion

Booking Details:

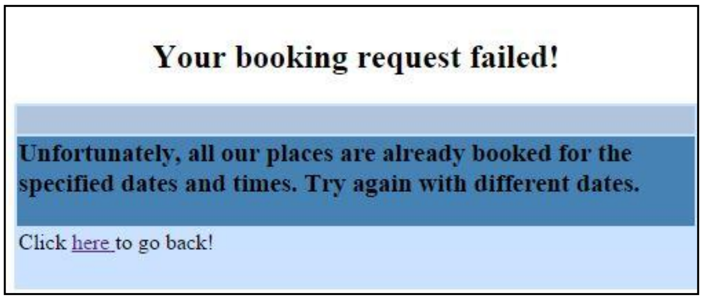
Arrival Time (DD/MM/YYYY - HH:MM):	15/04/2016 - 10:00
Departure Time (DD/MM/YYYY - HH:MM):	15/04/2016 - 11:00

If the car park has an available space on the specified dates and times, a confirmation screen should appear, or else, an error screen should be shown to the user. The two pictures below shows what to expect on both scenarios:


Booking successfully processed!

Your place was successfully booked at CarPark. Visit localhost:8081 in order to retrieve your eTicket.

Click here to go back!

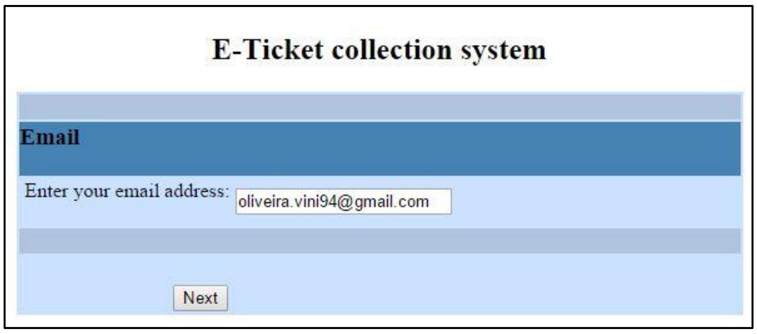

Your booking request failed!

Unfortunately, all our places are already booked for the specified dates and times. Try again with different dates.

Click here to go back!

3.3.2 ETicket retrieval

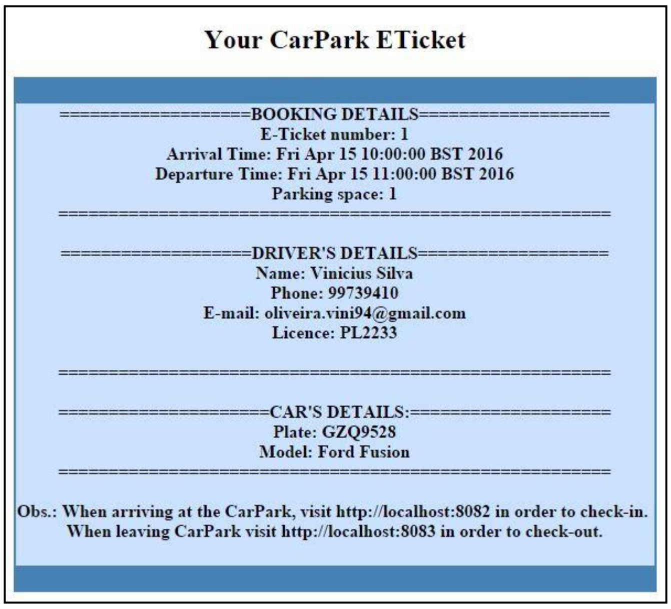
After typing "http://localhost:8081" on the browser address bar, the eTicket collection screen should be displayed. The user should then enter the same email used to book a place in the car park, and click on the "Next" button placed on the bottom of the page. The picture below shows an example of how it works.


E-Ticket collection system

Email

Enter your email address: oliveira.vini94@gmail.com

After entering an email address in the proper field, the user should see the eTicket displayed in the screen. If the entered email was not used to book a place in the car park, an error message is displayed. The two pictures below show what to expect on both scenarios:


Your CarPark ETicket

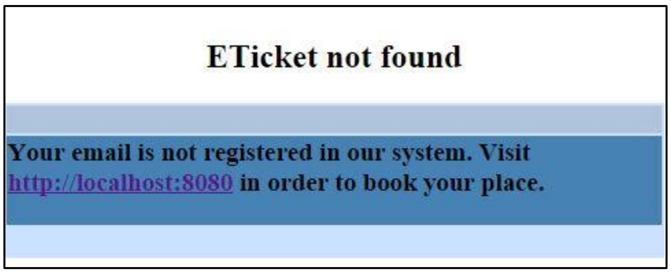
```
==================BOOKING DETAILS==================
E-Ticket number: 1
Arrival Time: Fri Apr 15 10:00:00 BST 2016
Departure Time: Fri Apr 15 11:00:00 BST 2016
Parking space: 1
==================================================

==================DRIVER'S DETAILS==================
Name: Vinicius Silva
Phone: 99739410
E-mail: oliveira.vini94@gmail.com
Licence: PL2233

==================================================

==================CAR'S DETAILS:==================
Plate: GZQ9528
Model: Ford Fusion
==================================================

Obs.: When arriving at the CarPark, visit http://localhost:8082 in order to check-in.
When leaving CarPark visit http://localhost:8083 in order to check-out.
```

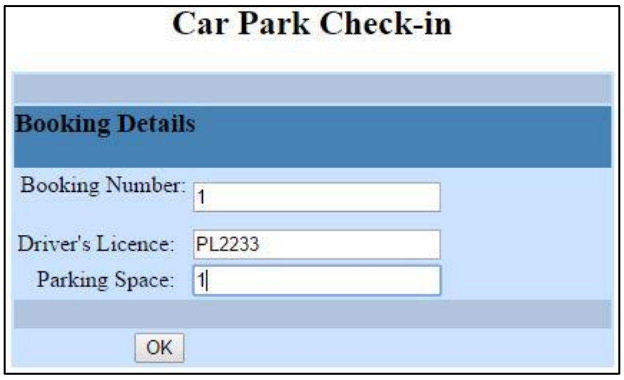

ETicket not found

Your email is not registered in our system. Visit http://localhost:8080 in order to book your place.

The user should also check the inbox of the informed email and verify if he / she has received a message from the system containing a copy of the eTicket. This is important because in order to check-in and check-out at the car park, the customer must present some information provided in the eTicket.

3.3.4 Car Park Arrival

In order to simulate the driver's arrival at the car park, the system provides a service on the computer's port 8082 that allows the users to enter some identification data and their eTicket number, enabling them to check-in at the car park. After typing "http://localhost:8082" on the browser address bar, the arrival screen should be displayed. The user should then provide the required information and click on the "OK" button placed on the bottom of the page. The picture below shows an example of how it works.

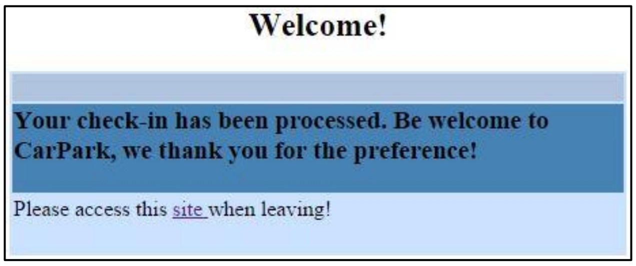


Car Park Check-in

Booking Details

Booking Number:	1
Driver's Licence:	PL2233
Parking Space:	1

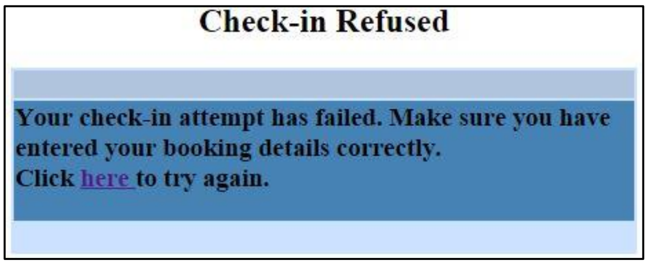
After filling the form and clicking the "OK" button, the user should see a welcome message informing that the check-in was successful. If the information provided is wrong, an error message is shown. The two pictures below show what to expect on both scenarios:



Welcome!

Your check-in has been processed. Be welcome to CarPark, we thank you for the preference!

Please access this site when leaving!

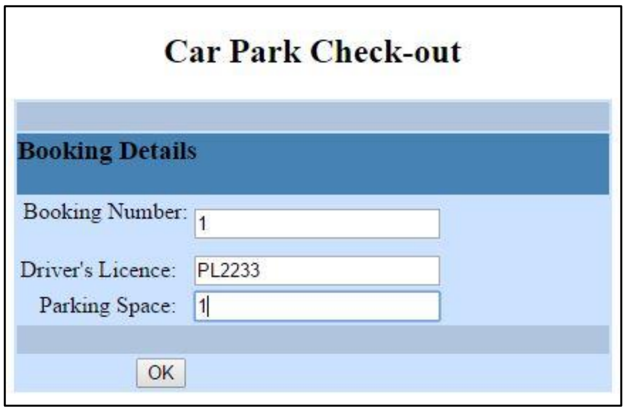


Check-in Refused

Your check-in attempt has failed. Make sure you have entered your booking details correctly.
Click here to try again.

3.3.5 Car Park Departure

In order to simulate the driver's departure from the car park, the system provides a service on the computer's port 8083 that allows the users to enter some identification data and their eTicket number, enabling them to check-out at the car park. After typing "http://localhost:8083" on the browser address bar, the departure screen should be displayed. The user should then provide the required information and click on the "OK" button placed on the bottom of the page. The picture below shows an example of how it works.

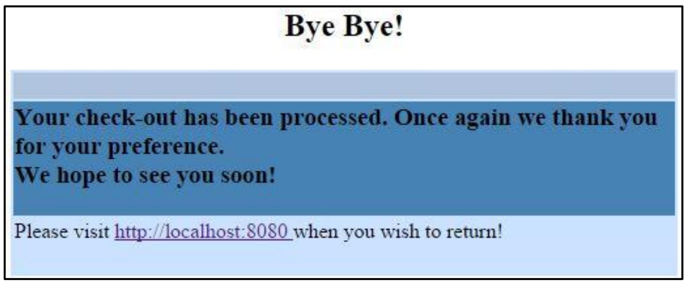


Car Park Check-out

Booking Details

Booking Number:	1
Driver's Licence:	PL2233
Parking Space:	1

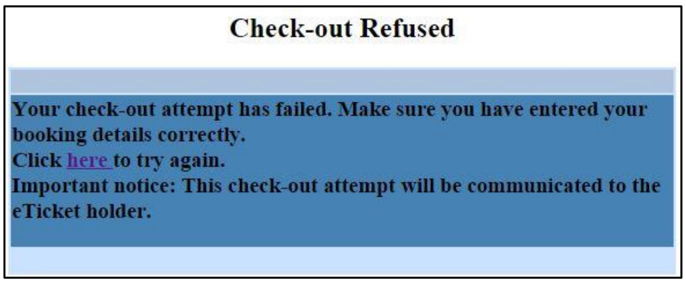
After filling the form and clicking the "OK" button, the user should see a farewell message informing that the check-out was successful. If the information provided is wrong, an error message is shown. The two pictures below show what to expect on both scenarios:



Bye Bye!

Your check-out has been processed. Once again we thank you for your preference.
We hope to see you soon!

Please visit http://localhost:8080 when you wish to return!



Check-out Refused

Your check-out attempt has failed. Make sure you have entered your booking details correctly.
Click here to try again.
Important notice: This check-out attempt will be communicated to the eTicket holder.

4. Solution Analysis:

In order to evaluate the correctness of the presented solution, a logging system was integrated to the system. A log file named "log.txt" is generated while the system is running and shows the traces of the execution of the components. By analysing this file, it is noticeable that the communication among the components is done in an alphabetized way. When the system is running, in order to show the synchronization, whenever a component writes or reads something in/from a channel, an entry is added to the log file. By behaving this way, we can expect the system to generate a log file in which each entry is duplicated (one entry for the 'speaker' and one entry for the 'listener').

When observing the log file, it is possible to see that indeed, the entries are duplicated, which indicates that the synchronization in the components communication is actually working. It is then possible to conclude that the alphabetized communication was really implemented in the system.
Another insight that the analysis of the log file provides is the idea of prefix closure. In every run of the system, the log file must show the events in the correct order so that we know there is no non-determinism in the system.
When there are multiple users using the system at the same time, it is possible that the log shows some events in an odd order, which seems to indicate non-deterministic behaviour. However, this fact is expected and does not mean that the system is behaving in a non-deterministic way, as the wrong-ordered events refer to different users and therefore, the system is working correctly.

5. Conclusion

The development of the presented coursework has occurred as expected, with no major issues. It is important to state that a deep learning in concurrent systems development was achieved during the execution of this project. The obtained results are consistent with the expectations and show that CSP is a powerful technology that has potential to significantly help programmers to overcome several problems that are inherent to concurrent systems development.
In general, it can be said that the coursework development was satisfactory even though the coursework specification was not very precise. Many assumptions were needed and intense use of creativity was necessary in order to design the presented model.
In conclusion, the process of development of the model was a very good opportunity for us to have a first contact with practical concurrent systems design using a non-conventional approach, which really contributed to our development as computer scientists.

6. References:

- *Communicating Sequential Processes for Java* (2014). Accessed on February 18, 2016, available at https://www.cs.kent.ac.uk/projects/ofa/jcsp.

- *Java Mail API*. Accessed on February 19, 2016, available at https://java.net/projects/javamail/pages/Home#Download_JavaMail_Release

- Belapurkar, Abhijit (2005). *CSP for Java Programmers, Part 1.* Accessed on February 18, 2016, available at http://www.ibm.com/developerworks/java/library/j-csp1/index.html

- Belapurkar, Abhijit (2005). *CSP for Java Programmers, Part 2.* Accessed on February 18, 2016, available at http://www.ibm.com/developerworks/java/library/j-csp2/

- Belapurkar, Abhijit (2005). *CSP for Java Programmers, Part 3.* Accessed on February 18, 2016, available at http://www.ibm.com/developerworks/java/library/j-csp3/

7. Appendix

The source code files that make up the system are available in the jar file included within the zip file that was uploaded along with this report. The .java files are organized in 3 packages: “csp”, “exceptions” and “model”. The html files that implement the GUI are also in the zip archive and it is important that when the system is run, these files be in the same folder as the executable jar.
Another important point to notice is that in order to the system to work, the computer must not be executing any services that use any of the 8080, 8081, 8082 and 8083 ports. An active connection to the internet is also required for the system to work properly (only the emailing extra functionality has this requirement).