# SOFTENG752 Assignment 2: akna890

## Part1: CSP

- USE implementation is located at file ***OnlineShop.use***, which contains the OCL specification of an online shop
- The test script of the USE implementation is located at file ***OnlineShop.soil***. You can load this test script by doing the following:
    1. Open the ***OnlineShop.use*** in the USE program
    2. Open the ***OnlineShop.soil*** by running ***open OnlineShop.soil*** in the USE CLI window
    3. From this point the test script should run and verify the classes, invariants and operations in the OCL program

Below is an image of the sequence diagram of the test case where a scenario of a user purchasing and returning items from an online shop.

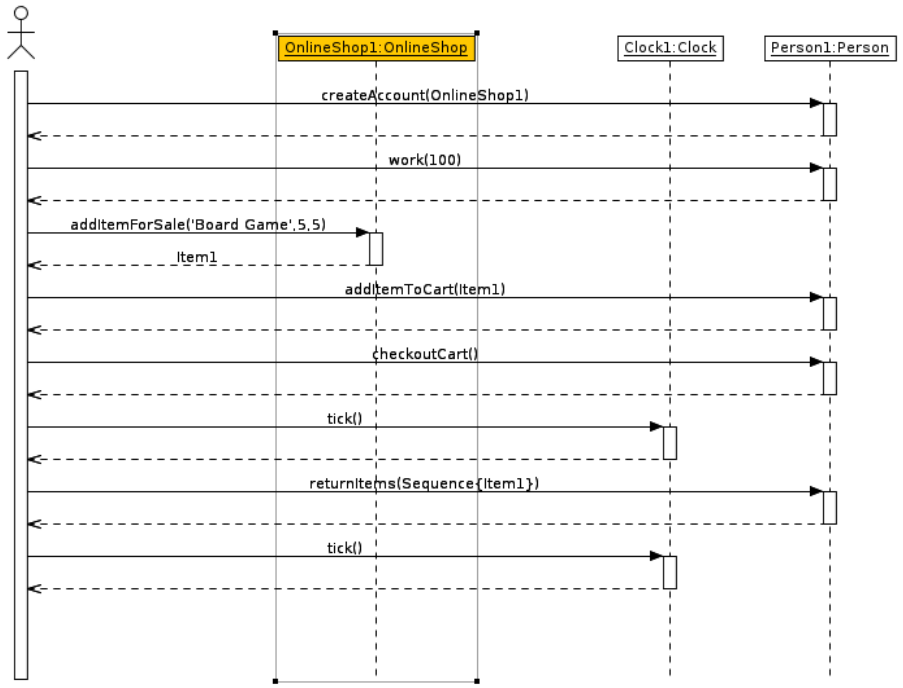


**Figure 1: *OnlineShop.soil*** test case sequence diagram for ***OnlineShop.use***

Explanation:

- User must first create an account with the shop
- Then they must work to earn some funds (100 dollars)
- The shop creates a new board game item and adds it for sale
- The user adds the board game item to their cart
- Then the user checks out the cart, paying for the item and getting it shipped
- After tick() (indicating some time has passed), the shipping items are delivered to the user
- The user then decides to return the item, shipping it back to the store and getting a monetary refund
- The tick() operation delivers the items back to the store

## Part2: CSP

- The CSP file ***OnlineShop.csp*** can be opened from ProB
- The model has two operations where a shop may add an item to sell (***addItemForSale***) which enables a user to purchase the item without an error occurring
- Whenever an error type occurs it is hooked (captured as an interrupt by ***KILLABLE_SYSTEM***) and the entire composed set of processes are stopped
- When a user purchases an item their balance (modelled by variable **X** in the ***USER*** process) decreases by one, and the user can return an item (***returnItem***) to increase their balance
- If the users balance is zero the system resets so that the user has a balance of 5
- All assertions pass in the ProB tool
- (Figure 2) shows that all assertions on the system pass/are correct

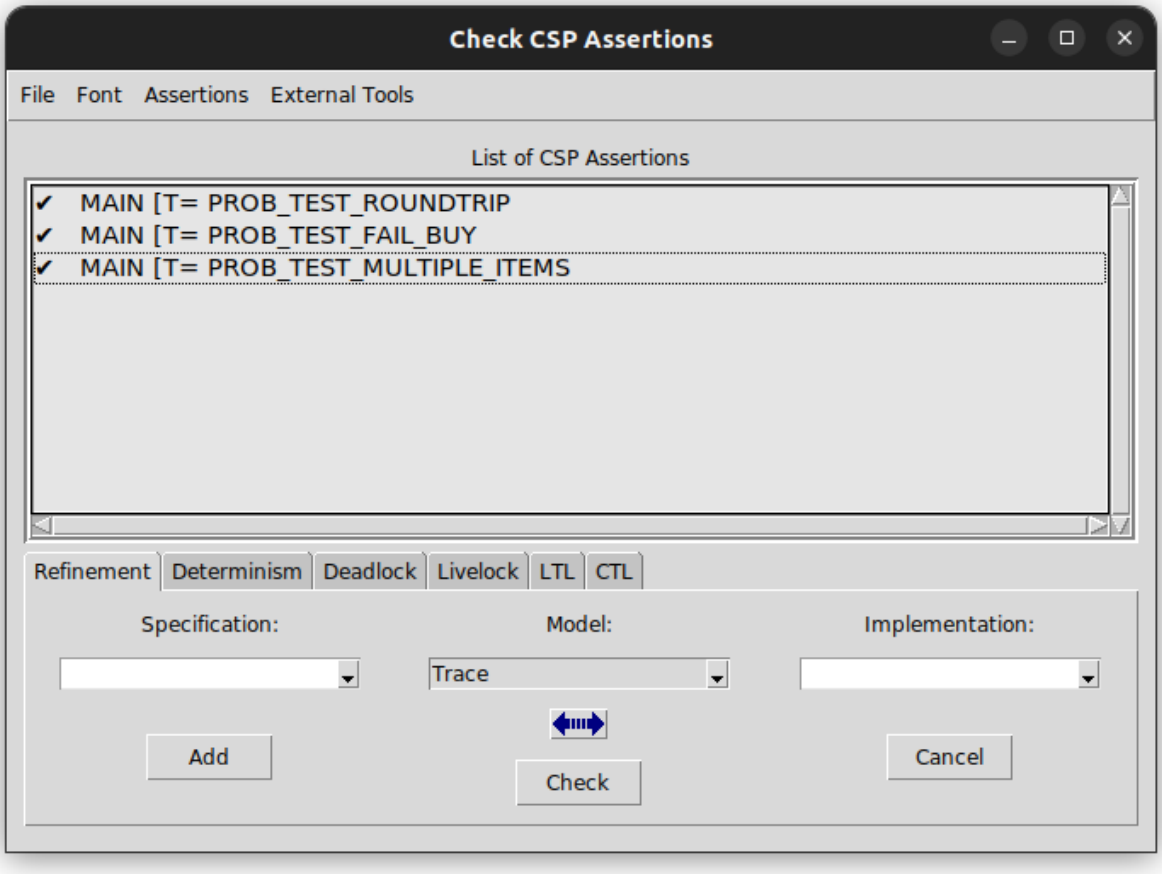


**Figure 2: *OnlineShop.csp*** assertion check passing

## Part3: Comparison of Formal Specification Modelling Languages

- Z and B allow for completely automated testing in which the entire state space is traversed and invariants are checked at every step of the process, this provides an easy (at least in manual human effort) method of completely testing a models correctness
- USEOCL and CSP are manually tested by the user. The user must create test scenarios (in OCL this is a .soil script, while in CSP the main process is checked

against assertions) in which the assertions/operations are tested against the invariants, preconditions and postconditions of the system. Generally testing in USEOCL/CSP require more manual effort

- Connections between Z, B, OCL and CSP is that Z provided a very basic design which did not have a shipping system, while in B a shipping system was implemented where items are shipping to/from the user within a day (has a clock functionality). CSP manages to implement all the features of the B system, including shop, user and shipping, buying and returning, balances for the user, etc. However the OCL design manages to create this system in far fewer lines of code than the B system. The OCL online shop explores the same relationships in the B design but uses multiple classes to do this which makes the relationship between different pieces of code clearer
- All Z, B, USE OCL and CSP offer some sort of hiding or code hierarchy. Z allows reuse via encapsulation of user defined classes which can be thought of as an extension to behaviour. B has the EXTENDS, USES, etc clauses which allow for reuse of machines and OCL has packages and extension of child/parent classes. OCL has composition and overriding of processes and events which can be thought of as an extension to a processes behaviour.
- The modelling differences between each system is that Z and B are declarative in nature, focusing on **what** is true about the system, while USEOCL is imperative, focusing on **how** the changes to the system are made. Overall the USEOCL design was easier to understand due to an extensive background knowledge in OOP.
- In Z and B, modelling of relationships is done by creating variables which are mutated by functions (or delta schemas in Z) which must respect invariants of the machine. However in USEOCL the relationships can be set more literally by establishing attributes between classes and setting invariants within each class, making the relationships between different abstractions in the code more visible and explicit.
- While each of Z, B and OCL are focused more on modelling the implementation of a system or a set of systems, CSP models the relationship between systems that communicate via events and the user environment (input/ouput). This is drastically different to Z, B and OCL as the implementation of a system is not the focus, rather CSP focuses on **how** each process (system) interacts with each other (rather than **what** the process represents). This means that implementation of process state (member variables) is generally not well supported in CSP, however how classes interact is most apparent/visible in CSP.
- On reflection I found that OCL was significantly easier to create models with, due to its expansive standard library and the ability to cast types (eg: cast a sequence to a bag or a set). For example, with the B language I was required to manually write a companion library for sequences, including a function to find the index of an element, while in USEOCL I didn't have to do this due to ***indexOf*** being built into the standard library.
- Additionally USEOCL has several functional features such as ***select***, ***type casting,*** and exlusion/inclusion predicates. These are functional programming features that make development significantly easier. In comparison mapping over a sequence in B was difficult because of WHILE loop semantics.
- However Z and B also have their advantages, I find that I was able to write terser code in Z and B due to it being declarative, which often made implementations easier

as I could just focus on what was true about the system and leave ProB to figure out what operations were necessary to achieve the statements I wrote, while in OCL I have to manually update the state of the system one step at a time via multiple functions to achieve a desired state

## Part4: Notes

- USEOCL refers the the USE flavour of OCL, which supports a subset of OCL syntax.