Specification Critique

Group 03
se2020.grp03@cse.unsw.edu.au

05 August 2012

Contents

1 Specification Critique

1.1 Introduction

Invariants & Theorems

Invariants should express all constraints on the machine that are important to the integrity of the system. They are not merely used as a method to declare variable types. The invariants should be used to specify the semantic relationships between variables. It is also said that invariants should be as strong as necessary but no stronger. Theorems provide a way for checks to confirm those properties that are *"obviously"* true.

Concrete

The models should describe behaviour, not details of how that behaviour is obtained. Therefore, models should be abstract, rather than concrete.

Machine Sequence

The way the refinements are carried out in the model should be in a way that the previous machine has a link to the next. The refinements should carry over properties is related to the previous machine. Machine Sequence affects the flow of implementation and deployment sequence.

Clarity

The model should be readable by humans. A model that looks complicated and impressive might not be a good model. Don't confuse complexity with quality. Readability of naming convention is also very important. Clarity will allow for easy implememtation.

1.2 Invariants & Theorems

Invariant used to reinforce semantic relationships

Group 4

Semantic relationship are reasonably obvious in group 4's model- there were strong constraints in some machines that show linkage between each variables. These constraints were primarily relationship between size and numbers, and were particularly stronger in areas such as inventory and warehouse stock management. Such example would include usedSize and maxSize from the storage area machine.

However in machines where integrity constraints were lackluster, it can be seen that this is due to the model's several 'small refinements', where only one to four variables were only introduced. This became apparent with the newer variables having no linkage with the former variables, such as warehouseThreshold and a storage's usedSize. Furthermore, basic number values and size comparisons were neglected, for example the relationship between the currentDay, a product's timeToExpire and its expiryDate, despite including many of these in the earlier refinements. These relationships were only reinforced in the guards in the relating events.

In one respect, the lack of constraints can be seen as acceptable, owing to the fact that this spec attempts to model real world restrictions through its idea of having general attributes, and essentially having one variable representing the many viable 'loss' constraints.

Group 7

Group 7's invariants and their properties are similar to group 4's, in that some constraints were included and appropriate at times, and lacking in the later refinements- many of the later invariants were type based constraints. It was also noted that this group's spec had the most number of refinements out of the four.

Similar to group 4, each of these refinements, especially the latter ones introduced one to three variables. There was also a distinct lack of important associations between older and newer variables, for example, the stock threshold in each floor and the floor's capacity as well as orders and their amount not exceeding the capacity. This can be largely due to high numbers of refinements, where the separated related variable become easily lost between each refinement.

The constraints were largely relevant to warehousestock, and floor locations, similar to group 4's, where it establishes separate compartments and entities a product has to exist in. For example, the linkage between products, stock, floor, backroom and warehouse in the location machine. Compartmentalisation is also seen with registers, in particular moneybox, storeSafe and sumOfRegisters, in the later refinements.

Group 11

Group 11 had both appropriate and unnecessary invariants. Given its high number of undischarged POs, it can be seen that many of these unnecessary invariants backfired and rooted logical and correctness issues. Similar both group 4 and 7, many of these integrity constraints were not sustained throughout the refinements, with lesser in the loyalty, discounts and schedule machines.

Integrity constraints were stronger in areas such as transactions, where linkage was very clear between how the instore cashtill operates with its topay and transactionInProgress variables, accompanied by comments that describes this relation. The spec also at times associates variables from its previous refinements, such as products in the shoppingcart with the number of stocks available to purchase.

There were no invariants in the Users machine where authorisation and privileges were enforced. Many of these relationships depended on its event based guards, with no invariant that ties together what a higher level access level can general 'modify'. The lack of constraints in the more complicated concepts such as the spec's scheduling system created unclear and ambiguous functionalities in the spec. For example, whether if a schedule can be both a specialSchedule and orderSchedule at the same time and function concurrently. Moreover, this ambiguity in the model underscores flaws in the original requirements and design, rather than just the spec/event b model itself.

Group 3

Based on the above three specs, we (group 3) derived a number comparable improvements to be made. The main issue with our spec concerns the lack of constraints that describes the semantic relationship between variables, and this can be seen through the number of proof obligations our model had in total. Through identifying and understanding other models, we were able to highlight how weak constraints impact the cohesion as a system, and the corrective value of the model.

It was also observed that while the above three specs operates with multiple functioning trolleys of products by different users simultaneously, our model handles one transaction at a time, given one moneybox and trolley, and this can somewhat infer a distinct relationship with the number of complex constraints our spec had to consider.

Despite having no integrity based constraints, some of our guards were sufficiently comprehensive, and thus can be translated and generalised to become more appropriate invariants. For example, the interaction between the purchases of ReservedStock, ReservedNum, stock and trolley with ActiveProd can be readily prepared and translated to an invariant based on the BuyReserve and ReserveProduct events. With these guards, our team can consider outlining possible and more effective invariants to add and improve in the later stages.

Use of theorems

In all four groups, no theorems were included, and will be disregarded for this basis of this critique.

1.3 Concrete

Group 4

This group demonstrated good abstract design for most of their requirements. As we can see, the product requirements only describe the behaviour of the system instead of depicting specific stuff on how the requirement will be accomplish. Looking at PD–2.2.1 and PD–2.2.2 (appendix A, figure 1), the two product requirements have goo abstract design because they just model the behaviour of the system without providing any specific ways or information that suggests how the system would meet the goal. However in event-B, their way of constructing the user access is too concrete because they have to specify which user was granted the access in each function. Also some of the events contain too many details. For example in the MoveStock event (appendix A,figure2), this event contains too many guards that would make this event too specific. The group should distribute the guards out of the machine would be more abstract.

Group 7

Overall, this group have most of the requirements well model, but a few requirements are too specific and can be broken down further. For instances, PD–2.5.5 (appendix B,figure1) is too specific because I know how the goal would be accomplish. They should move those requirements down or make them more abstract. For example, PD–2.5.3(appendix B,figure1) should be replaced as the system can handle item storage so then in the design level can

then state that the same items would be place at the same location. As for the Event-B, they have too many guards in each function that would make their model too specific. For instance in the event AddStockToWarehouse (appendix B,figure2), the function has a lot of guards while there are only two actions. Some of those guards can actually be moved to the invariants at each machine instead of within each function. Then in the BuyTrolly event (appendix B,figure3), the function contains too many details which would make the model too concrete. Like in a scenario when the registrar ran out of money, the guard would actually restrict the staff from making the transaction because the guard states it so therefore the event can not be fired. Maybe the cashier would actually want to settle the transaction first then proceed to get the changes for the customer. An abstract event would need to be as general as possible to compute with as many situations as possible.

Group 11

This group has the most concrete requirements among the four groups. Referring to PD–1.2.6 (appendix C,figure1), the requirement is too concrete because I can figure out how the goods will be dealt with when it is in a replacement order. The requirement should be more abstract like the system can handle replacement orders or else you can't extend that product requirement further down to design level. Then the event-B for this group has a lot of specific events that probably would only suit for some limited situation. By taking a look at this simple event Give Change (appendix C,figure2), this event has too many guards for only one operation. More guards should be delegated at the top of the machine and the invariants could be more general to make the model more abstract.

Group 3

This group has most of the requirements model fairly abstract with a few requirements that could be better. For example PL–2.4.3 (appendix D,figure1) contains specific information that list out the attribute that the system would store. The requirement can be replace as the system can record product details and then extends that requirement down to design level where it can then state the previous attributes that were mentioned. Most of the events in Event-B were modelled well except for some events like AddProductStore (appendix D,figure2), the event contains too many guards that would make the function too specific. The function could be better to rearrange the guards out of the functions and also reduce the amount of information in the event.

Conclusion

Overall, most of the groups have their requirements well modelled, but the Event-B was too specific in general. Out of all the groups, group 4 have the most abstract model for both the requirement and Event-B. Our group (group 3) is place in the middle because most of our requirements are well modelled. Group 11 have most concrete model and contains too many details in some of the functions making it cause too many errors in the model.

1.4 Machine Sequence

Group 04

Machine Sequence

StorageArea –> WarehouseR0 –> WarehouseR1 –> InventoryR3 –> StoreR4 –> StoreR5 –> StoreR6 –> AdministrationR7–> WarehouseR8 –> StoreR9 –> CustomerR10 –> ExpiryR11 –> ExpiryR12 –> SplittersR13 –> ExpiryR14

Critique

Group 04 has the most number of machines among the 4 groups that we have decided to critique on. Coming in at a total of 15 machines. They started their model well. As the machines have strong relationships/linkage between each other as the refinement progresses. But started showing problems in their from AdministrationR7 onwards. Having not planned well before hand, they had to refine AdministrationR7 into WarehouseR8 to add in *event SetThreshold* and *event MoveStock refines AddObject*. They could have easily done that in StoreR6.

Following on, they modelled StoreR9 to be able to choose, purchase, refund and move products. CustomerR10 to model loyalty points and memberships. Next, they followed on by modelling ExpiryR11, ExpiryR12. These refinements could have been incorporated into previous machines and clearly do not have any relations to the previous machine CustomerR10.

They tried to split the stores/warehouse into different areas with the next refinement being SplittersR13. And with the new areas, they had to set the expiry to move stock to the new designated areas resulting in the refinement ExpiryR14. Due to lack of foresight and planning, the group did not have a good machine sequence.

Instead of continuing to refine the machines in this way, they could have saved themselves much trouble by actually editing the previous machines to reflect the changes in work flow instead of pressing on.

Group 07

Machine Sequence

POS-Product-R0 –> POS-Stock-R1 –> POS-Location-R2 –> POS-Trolley-R3 –> POS-User-R4 –> POS-Return-R5 –> POS-StorageCapacity-R6 –> POS-UserClasses-R7 –> POS-Days-R8 –> POS-Threshold-R9 –> POS-Registers-R10 –> POS-ProductType-R11 –> POS-Discount-R12 –> POS-PaymentMethods-R13

Critique

Group 07 has quite a number of machines, 14 machines. Although unlike Group 04, overall, they have a better thought out model in terms of machine sequence. Most of the refinements are clearly related with each refinement building on to the previous machine. Improvements could still be made though.

If they had added in the different user capabilities from POS-UserClasses-R7 into POS-User-R4, they could have brought POS-Threshold-R9 and POS-ProductType-R11 up into the earlier refinements after POS-StorageCapacity-R6 and the later refinements would only be dealing with *Registers, Discounts and Payment Methods* which would have a better flow. In this case, even when dealing with a large number of machines, Group 07 has done well in terms of machine sequencing.

Group 11

Machine Sequence

BasicPOS –> Orders –> Locations –> Users –> Transactions –> Loyalty Program –> Price Modification and Refund –> Schedule

Critique

Group 11 has less machines compared to the previous 2 groups. The flow is similar to group 07, with the basic warehousing system modelled first, followed by ordering of stock from suppliers. And then on to the Locations which should come before Orders, as it makes for a better machine sequence. The next refinement, Users and then to Transactions followed by the Loyalty Program. Price Modification and Refund machine should come before the Loyalty Program as it has more to deal with Transactions. Schedule should come after Orders or Locations as it deals with the scheduling of stock and not refined from Price Modification and Refund

Group 03

Machine Sequence

PoSWare –> WarehouseR1 –> StoresR2 –> RetailR3 –> MembershipR4

Critique

Our group also has less machines compared to group 04 and group 07. Comparatively, our group has good machine sequencing. With modelling the behaviour of the overall system in PoSWare and refining it into Warehouses and Stores. Following that, it was further refined to Retail to deal with transactions and refunds. And the finally refined into Membership to deal with various membership benefits.

1.5 Clarity

Naming the project

The name of project should clarify breafly at the first look.In SENG projects, It's a good practise to mention the Group number and the module's name which is moduled. None of the group have done it properly, Group 07 chose the name "SENG", Group 04 chose "PosWare" and Group 11 called it "POS". None of these gives any information about which group are these which just make it hard for marker to figure group's number by himself. However group 3 just called Group3, which is the best in here but not enough. (no mentioning the modules name)

Good use of naming varables

Group 4

At early stage of refinement group 4 have clear and distinguished names for variables and parameters, but at the end many names are almost repeated for different varable without extra explanation such as : “basket objects” and “basket product”.

Group7

Chose resemble naming for their variable but they haven’t keep the correct style of naming system, such as instead of “floorShelves” they used “floorshelves” which in a long list will makes hard to distinguished.

Group11

Group 11, made the same mistake as Group 7. such as “pointsaccumulated” , These mistakes makes it hard to read, which is one of the main purposes of moduling.

Group3

Same mistake but not as much as other groups . As an example “Reorderlevel” Good naming of refinement and correct positions for refinement, for easy finding. In General a Week point of Event-B is it will order the machines based on alphabetic order, while it might shows notting and just caused the misunderstand. But having said that it can be handled. So reader can understand which machine is the refinement of which one.

Naming of Machines

Group 4

Group 4 did a trick by adding R[number] at the end of each machine and context . It number indicate the number of the refinement. However, they haven’t done it properly by forgotting to use this system for all machines , such as example “storage area” has no R at the end. Beside the ordering is still confusing . They could use the numbering at the beginning.

Group 7

Group 7 did the same technique but correctly , however they are also having the same ordering issue , including the extra “POS_” at the beginning which again make it harder to read trough the list. Such as “POS_Discount_R12”,”POS_Days_R8” and “POS_Return_R5”. It’s a psychological reason that eye first look at the start of line which here it’s just lead to confussion.

Group 11

This group haven’t done anything to solve this problem and it will make the reader just to look each machine and flow the path backward to reach the beginning. And again search for the refinement. So imature.

Group 3

This group haven't done anything extra than groups 4 and 7.

Conclusion: In order to make it easier to read the machines in project and find out the refinement of a machine, machines should be called 1.[machine_name], based on their ordering system, so evenB order it automatically and also the user knows which machine is the next refinement.

Adding new events properly

In EvenB when new events are added to the refinement they should be added to the top of the list, not at the button. This simple mistake will make the user just scroll down in each refinement to reach the new events.As an example the in group4 for machine ExpiryR12 to reach the new events user needs to pass 14 old events.The only group who did this properly is the group 3.

To sum it up, some easy practice which can make the module easier to read have been forgotten by many groups. Such as mentioning the group number and the module name on project name. Clear and distinguished naming for variable events and parameters with using the correct naming style. Ordering the refinement in-order to make it easy to find the next refinement. And the last adding new events to the top of old event in order to prevent scrolling all the way down.

2 "New" Specification Summary

3 Project Plan

4 Appendix

4.1 A

Figure1

PD–2.2.1 (!M) System will Keep Track of all Transactions. All transactions made by the system will be stored in the system. PD–2.2.2 (!M) System will Keep Track of Related Transactions. Any related transactions are flagged in the system.

Figure2

\EVT {MoveStock}\cmt{ \\hspace*{2.4 cm} [PD–3.2.2 System will Automatically Order Stock from Back Store Room to Replenish Store Shelves] \\hspace*{2.2 cm} [PD–3.2.3 System will Automatically Order Stock from Warehouse to Replenish Back Store Room] \\hspace*{2.2 cm} [PD–3.2.4 System will Automatically Order Stock from Suppliers to Replenish Warehouse] } \EXTD {MoveStock} \begin{description} \AnyPrm \begin{description} \ItemX{user } \ItemX{toLocation } \ItemX{fromLocation } \ItemX{amount } \ItemX{product } \ItemX{objectSet } \Item{store } \end{description} \WhereGrd \begin{description} \nItemX{ grd5 }{ amount = 1 } \nItemX{ grd1 }{ user \in managers } \nItemX{ grd2 }{ toLocation \in storages

*} \nItemX{ grd3 }{ fromLocation \in storages } \nItemX{ grd6 }{ product \in productLines } \nItemX{ grd7 }{ storageProducts(fromLocation)(product) \geq amount } \nItemX{ grd8 }{ product \in dom(storageProducts(toLocation)) } \nItemX{ grd9 }{ product \in dom(storageProducts(fromLocation)) } \nItemX{ grd10 }{ fromLocation \neq toLocation } \nItemX{ grd11 }{ maxSize(toLocation) \geq usedSize(toLocation) + (amount * size(product)) } \nItemX{ grd12 }{ warehouseThreshold(toLocation)(product) \geq storageProducts(toLocation)(product) } \nItemX{ grd13 }{ objectSet \subseteq objects } \nItemX{ grd14 }{ objectSet \subseteq productObjects(product) } \nItemX{ grd16 }{ objectSet \subseteq storageObjects(fromLocation) } \nItemX{ grd15 }{ card(objectSet) = amount } \nItemXY{ grd17 }{ objectSet \nsubseteq storageObjects(toLocation) }{* \\hspace*{1.6 cm} shouldn't be a problem } \nItemY{ grd18 }{ (fromLocation \in warehouseAreas \land toLocation \in backRoomAreas \\hspace*{1.4 cm} \land toLocation \in storeBackRooms(store)) \lor \\hspace*{1.4 cm} (fromLocation \in backRoomAreas \land fromLocation \in storeBackRooms(store) \\hspace*{1.4 cm} \land toLocation \in shelfAreas \land toLocation \in storeShelves(store)) }{* \\hspace*{1.6 cm} either moving from a warehouse to a store \\hspace*{1.4 cm} or from store backroom to it's shelf } \end{description} \ThenAct \begin{description} \nItemX{ act1 }{ storageProducts := storageProducts \ovl \\hspace*{1.2 cm} { fromLocation \mapsto { product \mapsto (storageProducts(fromLocation)(product)- amount)} , \\hspace*{1.2 cm} toLocation \mapsto { product \mapsto (storageProducts(toLocation)(product)+amount)} } } \nItemX{ act2 }{ usedSize := usedSize \ovl \\hspace*{1.2 cm} { fromLocation \mapsto (usedSize(fromLocation) - (amount * size(product))), \\hspace*{1.2 cm} toLocation \mapsto (usedSize(toLocation) + (amount * size(product)))} } \nItemX{ act3 }{ storageObjects := storageObjects \ovl \\hspace*{1.2 cm} { fromLocation \mapsto (storageObjects(fromLocation) \setminus objectSet), \\hspace*{1.2 cm} toLocation \mapsto (storageObjects(toLocation) \bunion objectSet)} } \end{description} \EndAct \end{description}*

4.2 B

Figure1

PD–2.5.3 Items are placed under the same classification and this will define which shelf it is placed on PD–2.5.5 The placement of a product on a specific shelf is defined by the size of the store, it's location and season

Figure2

\EVT {AddStockToWarehouse} \EXTD {AddStockToWarehouse} \begin{description} \AnyPrm \begin{description} \ItemX{product } \ItemX{amount } \ItemX{manager } \ItemX{expiryDay } \end{description} \WhereGrd \begin{description} \nItemX{ grd1 }{ product \in products } \nItemX{ grd3 }{ product \in dom(warehouse) } \nItemX{ grd2 }{ amount \in 0\upto warehouseLimit } \nItemX{ grd4 }{ disabled(product) = FALSE } \nItemX{ grd50 }{ warehouse(product) + amount \leq warehouseLimit } \nItemX{ grd5 }{ currentWarehouseCapacity + amount \leq maxWarehouse } \nItemX{ grd7 }{ manager \in employees } \nItemX{ grd6 }{ employeeGreaterThanOrEqualTo(employeeClasses(manager) \mapsto MANAGER) = TRUE } \nItemX{ grd8 }{ activeEmployees(manager) = TRUE } \nItemX{ grd9 }{ expiryDay \in 0\upto maxDays } \nItemX{ grd10 }{ expiryDay \geq day } \nItemX{ grd11 }{ dayStarted = TRUE } \end{description} \ThenAct \begin{description} \nItemX{ act1 }{ warehouse(product) := warehouse(product) + amount } \nItemX{ act2

}{ currentWarehouseCapacity := currentWarehouseCapacity + amount } \nItemX{ act3 }{ productExpiry(product) := expiryDay } \end{description} \EndAct \end{description}

Figure3

\EVT {BuyTrolley} \REF {BuyTrolley} \begin{description} \AnyPrm \begin{description} \Item{payment } \Item{user } \Item{staff } \Item{register } \end{description} \WhereGrd \begin{description} \nItem{ grd3 }{ payment \in 0\upto maxPrice * maxAmountToBuy } \nItem{ grd2 }{ user \in users } \nItem{ grd6 }{ activeUsers(user) = TRUE } \nItem{ grd1 }{ payment \geq costOfTrolleys(user) } \nItem{ grd4 }{ \forall p \qdot p \in products \\hspace*{1.2 cm} \limp productsInTrolleys(p) \geq trolleys(user)(p) } \nItem{ grd5 }{ \forall p \qdot p \in products \\hspace{1.2 cm} \limp p \in dom(productsInTrolleys) } \nItem{ grd10 }{ userGreaterThanOrEqualTo(userClasses(user) \mapsto CUSTOMER) = TRUE } \nItem{ grd7 }{ staff \in employees } \nItem{ grd8 }{ employeeGreaterThanOrEqualTo(employeeClasses(staff) \mapsto STAFF) = TRUE } \nItem{ grd9 }{ activeEmployees(staff) = TRUE } \nItem{ grd11 }{ user \neq staff } \nItem{ grd12 }{ dayStarted = TRUE } \nItem{ grd13 }{ register \in registers } \nItem{ grd14 }{ registerMoney(register) + costOfTrolleys(user) \leq maxRegister } \end{description} \ThenAct \begin{description} \nItem{ act1 }{ registerMoney(register) := registerMoney(register) + costOfTrolleys(user) } \nItem{ act3 }{ trolleys(user) := emptyTrolley } \nItem{ act2 }{ costOfTrolleys(user) := 0 } \nItem{ act4 }{ totalCostOfTrolleys := totalCostOfTrolleys - costOfTrolleys(user) } \nItem{ act5 }{ productsInTrolleys :| \forall p \qdot p \in products \land productsInTrolleys \in products \tfun \nat \\hspace*{1.2 cm} \limp productsInTrolleys'(p) = productsInTrolleys(p) - trolleys(user)(p) } \nItem{ act6 }{ pastPurchases(user) := userCurrentPurchases(user) } \nItem{ act7 }{ currentFloorCapacity := currentFloorCapacity - numProductsInTrolley(user) } \nItem{ act8 }{ numProductsInTrolley(user) := 0 } \nItem{ act9 }{ sumOfRegisters := sumOfRegisters + costOfTrolleys(user) } \end{description} \EndAct \end{description}

4.3 C

Figure1

PD–1.2.6 Goods that have been replaced by a replacement order are automatically placed into a special/spot sale specification. Replaced goods that are approaching use-by date will be placed into a discount class for spot sales and specials.

4.4 Figure2

\EVT {Give∼Change} \begin{description} \AnyPrm \begin{description} \Item{transaction } \Item{user } \end{description} \WhereGrd \begin{description} \nItem{ grd1 }{ transaction \in transactions } \nItem{ grd2 }{ transaction \notin transactionInProgress } \nItem{ grd3 }{ user \in users } \nItem{ grd4 }{ usergroups(user) \in { CLERK, MANAGER} } \nItem{ grd5 }{ change(transaction) > 0 } \end{description} \ThenAct \begin{description} \nItem{ act1 }{ change(transaction) := 0 } \end{description} \EndAct \end{description}

4.5 D

Figure1

PL–2.4.3 The system will record details about specific items (price, desc, weight, size, barcode)

Figure2

\EVT {AddProductStore}\cmt{ \\hspace*{3.6 cm} Adds a product to a store catalouge (sets quantity to 0) \|hspace*{4.4 cm} Note it only sets the stock for the back of the store. \\hspace*{4.4 cm} The reason is because a store may accepct refunds for an item that is not sold there } \EXTD {AddProductStore} \begin{description} \AnyPrm \begin{description} \ItemXY{item }{Any item } \ItemXY{store }{Any store } \ItemXY{stock }{Any stock } \end{description} \WhereGrd \begin{description} \nItemXY{ grd1 }{ item \in Product }{ \|hspace*{1.4 cm} The item is in the networked registery of Products } \nItemXY{ grd2 }{ store \in Store }{ \\hspace*{1.4 cm} The store is in the network of stores } \nItemXY{ grd3 }{ store \notin Warehouse }{ \|hspace*{1.4 cm} The store isn't a Warehouse (this is only for adding an item to a store) } \nItemXY{ grd5 }{ ActiveProd \bunion { store \mapsto item} \in Location \rel Product }{ \\hspace*{1.4 cm} The store mapped to an item is a possibility (PO) } \nItemX{ grd8 }{ StoreArea(store) = Backstore } \nItemXY{ grd6 }{ { store \mapsto item} \nsubset ActiveProd }{ \|hspace*{1.4 cm} The item is not currently being sold at the store } \nItemXY{ grd7 }{ stock = 0 }{ \\hspace*{1.4 cm} We are setting the initial quantity to 0 - incase we just want to add it to the store catalouge } \end{description} \ThenAct \begin{description} \nItemXY{ act1 }{ ActiveProd := ActiveProd \bunion { store \mapsto item} }{ \|hspace*{1.4 cm} The item is now in the catalouge of the store } \nItemXY{ act2 }{ StoreAreaItemQuantity(store \mapsto Backstore \mapsto item) := stock }{ \\hspace*{1.4 cm} We are setting the initial quantity to 0 - incase we just want to add it to the store catalouge } \nItemXY{ act3 }{ Stock(store \mapsto item) := stock }{ \|hspace*{1.4 cm} The total stock in the store (set equal to 0) } \end{description} \EndAct \end{description}