一個奠基於 UML 及 ASML 的測試神諭

魏啟仁　　林迺衛
國立中正大學　資訊工程學系
{wcj95m,naiwei}@cs.ccu.edu.tw

摘要

確認軟體是否符合軟體規格是一個困難的問題。其中一個困難是如何決定軟體的預期輸出。本論文將 Unified Modeling Language 的循序圖和狀態機圖的子集合及 Abstract State Machine Language 整合成為一個可執行的規格語言。本論文也開發一個此可執行規格語言的模擬器。此模擬器可以用來產生符合規格的軟體的預期輸出。此模擬器也可以用來實證規格是否符合客戶的需求。

關鍵詞：Unified Modeling Language，Abstract State Machine Language，測試神諭。

1 導論

本論文希望能提出一個模擬器可以較有效率的找出測試神諭(Test Oracle)所要用到的預期輸出。測試神諭屬於軟體測試中的一部分，它所負責的是檢查軟體在測試中所產生的實際輸出跟預期的輸出是不是相同。Unified Modeling Language(UML)是一個開發大型軟體專案時常使用的一種圖形化語言，UML 圖形上加一些規則及 Abstract State Machine Language(ASML)規格語言後就可以做為本論文所提出的模擬器的輸入，然後模擬器就可以依此產生預期的輸出。

本論文提出一個以 UML 圖形與 ASML 規格做為輸入的模擬器，此模擬器則可以告訴使用者按照所給定的規格所實做出的軟體會產生怎樣的輸出。本論文的貢獻有以下幾點：

第一，提出了一個可以將循序圖、狀態機圖及 ASML 規格結合在一起執行的模擬器。

第二，使得開發團隊可以在軟體開發的前期就知道所給定的規格是否是正確的，在軟體開發流程中越早發現問題越能減少軟體開發時所花費的成本及時間。

第三，因為本模擬器可以接受測試輸入並產生預期輸出，所以開發團隊在做軟體測試時只需要再考慮測試輸入的產生，也因此大幅降低了做軟體測試時的成本，這可以讓軟體開發團隊做較大量的測試以確保軟體的品質。

下面這張圖描述了模擬器的輸入及輸出的關係，循序圖(Sequence Diagram)、狀態機圖(State Machine Diagram)及狀態機圖上所附的 ASML 程式碼會輸入到模擬器上，而模擬器的輸出則是以這些 UML 圖做為規格所開發出的軟體應該會產生的預期輸出。

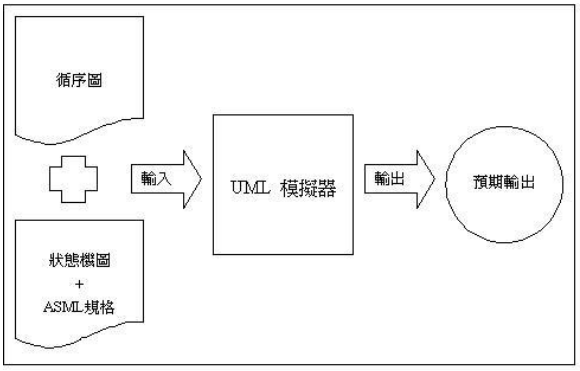

圖 1.模擬器輸入及輸出關係圖

UML 模擬器中含有循序圖模擬器、狀態機圖模擬器及 ASML 模擬器這幾個重要的元件。

因為循序圖模擬器負責模擬循序圖，而循序圖描述了軟體的主要流程，所以循序圖模擬器也控制了整個模擬的主要流程，此外循序圖模擬器也負責了從訊息的來源物件的狀態機圖中要求變數的值，以及將變數的值傳送給訊息的目標物件的狀態機圖中。

狀態機圖模擬器模擬循序圖所給定的觸發事件(Event)、檢查成立條件(Guard)以及執行活動(Activity)，檢查成立條件及執行活動這兩個行為

的模擬是將狀態機圖上有關的ASML規格傳給ASML模擬器去處理，此外，當循序圖模擬器向狀態機圖詢問某個變數的值之後，狀態機圖也會向ASML模擬器詢問該變數的值，並回傳給循序圖模擬器。

ASML模擬器是UML模擬器跟ASML編譯器之間的橋樑。狀態機圖模擬器會將狀態機圖上的成立條件及活動傳送給ASML模擬器，ASML模擬器在做完適度的修改後就會要求ASML編譯器去編譯出執行檔，而後ASML模擬器再去讀取此執行檔的輸出，並對ASML模擬器內的變數的值做修改，以供狀態機圖模擬器來詢問該狀態機圖內某個變數的值。

下面這張圖描述了UML模擬器的內部架構圖。

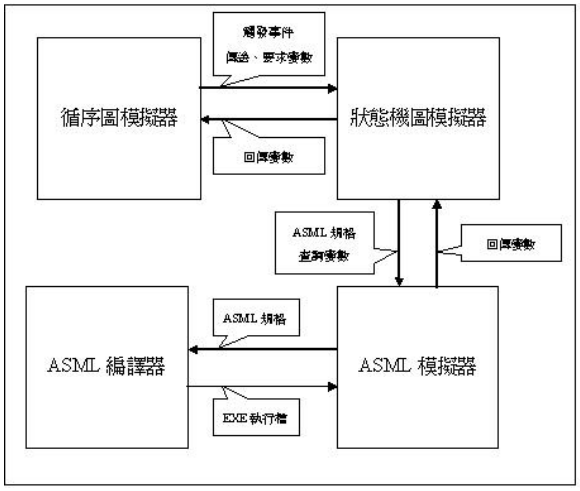

圖 2.UML 模擬器內部架構圖

本論文以下章節簡述如下：第二節將回顧相關研究。第三節簡介規格語言。第四節簡述一個範例。第五節詳述UML模擬器。最後，第六節總結本論文。

2 相關研究

軟體開發團隊在開發前期會先為要開發的軟體寫出規格，而這規格是實際撰寫程式碼時的根據。可執行規格語言則代表這個語言所寫出的規格是可以被直接執行的。本論文所提出的模擬器的輸入就是由ASML以及UML Diagram所結合而成的可執行規格語言，而接下來所要介紹的 Executable UML也是屬於規格語言的一種。

Executable UML是將UML去掉一些不可執行表示法後所成的子集合，而其目的是使軟體開發團隊可以定義出一些可執行的領域(domain)，而這些領域則描述了整個開發的系統。Executable UML目前還沒有一個完全公認的標準，為了利於介紹及解釋，接下來的介紹就以 Executable UML——模型驅動架構入門所提的內容做為標準[1]。

一個系統是由多個領域所組成，領域內的每個元素都有一個清楚的任務目標，例如使用者介面這個領域的目標就是跟使用者做互動。描述一個領域需要類別圖(Class Diagram)、狀態圖(State Diagram)及動作語言(Action Language)這三個元素。類別圖的描述與一般的UML圖形大致相同，而在狀態圖中則可能會以動作語言去描寫細部的行為，動作語言並沒有限定是哪種語言，不過若想使用某個Executable UML工具的話則必須使用該工具所規定的動作語言，若將本論文與 Executable UML做類比的話，則本論文的模擬器所使用的動作語言是ASML。

Executable UML 與本論文所提出的模擬器的目標類似，不過做法上略有不同，Executable UML是利用狀態機圖上的動作語言互相傳送觸發事件來執行整個流程，而本論文則是利用循序圖上的訊息做為觸發事件進而執行整個流程。如果有一個案例會需要狀態機圖間頻繁傳送觸發事件，照Executable UML的方法繪圖則可能很難去看出觸發事件的先後關係及各狀態機圖的關聯，而照本論文提出的方法繪圖則會得出一個較為龐大的循序圖，但這個循序圖還是可以描述出觸發事件的先後順序以及各物件之間的關聯。另外，若以Executable UML做類比則本模擬器所使用的動作語言是ASML。

測試神諭(Test Oracle)是一種檢查軟體在測試案例下是否有正確執行的機制。測試神諭分為神諭資訊(Test Oracle Information)及神諭程序(Oracle Procedure)，神諭資訊所指的是在某個測試案例下待測軟體的預期輸出，而神諭程序的工作則是將預期的輸出跟待測軟體真正的輸出去做比對，測試案例跟測試神諭合在一起就可以做一個完整的軟體測試了。Executable UML、本論文所提的

模擬器以及接下來要介紹的Automated Test Oracle Based On Neural Networks[1]都可以做為產生神諭資訊的手段。

Automated Test Oracle Based On Neural Networks[4]中使用類神經網路(neural network)產生預期輸出的近似值。類神經網路是機器學習(machine learning)的一個部分，機器學習的特色是先使用一些輸入跟輸出的結果訓練出一個系統，之後這個系統就可以對輸入做某種程度的輸出預測。神經網路先以給定的軟體輸入跟軟體輸入做訓練，訓練完之後若將測試輸入交給此神經網路處理，此神經網路就會產生一個近似的預期輸出。在已經擁有預期輸出後，第二步要做的就是對預期輸出與軟體實際的輸出做比較，因為類神經網路所產生的預期輸出是一個近似值，所以在Automated Test Oracle Based On Neural Networks中要求使用者給定一個精確度等級(precision degree)，接著計算預期輸出與實際輸出之間的差值的絕對值，當這個絕對值大於精確度等級時則認為軟體的輸出有問題，如果此絕對值小於精確度等級則認為軟體的輸出沒有問題。

3 規格語言

本論文所提出的模擬器所能執行的規格語言是由 UML 循序圖、UML 狀態機圖及 ASML 程式碼三個部分所組成的。

UML是Unified Modeling Language的縮寫，因為本論文只有使用到UML中的循序圖(Sequence Diagram)和狀態機圖(State Machine Diagram)，所以在此僅介紹這兩種圖[2]。

下圖是一個本篇論文所提出的模擬器可以模擬的循序圖。這是一個很簡單的循序圖圖，圖上表示的流程如下，buyer跟seller下訂單，seller啟動machine，seller要求machine開始生產物品，machine再告訴seller這筆貨物的原始價格，seller再告訴buyer這筆貨物的最後價格。循序圖上有多訊息在做傳遞，而訊息的來源方稱做來源參與物件，訊息的目標方被稱做目標參與物件。本論文所提出的模擬器對循序圖做訊息傳遞的格式做了限制，其格式為

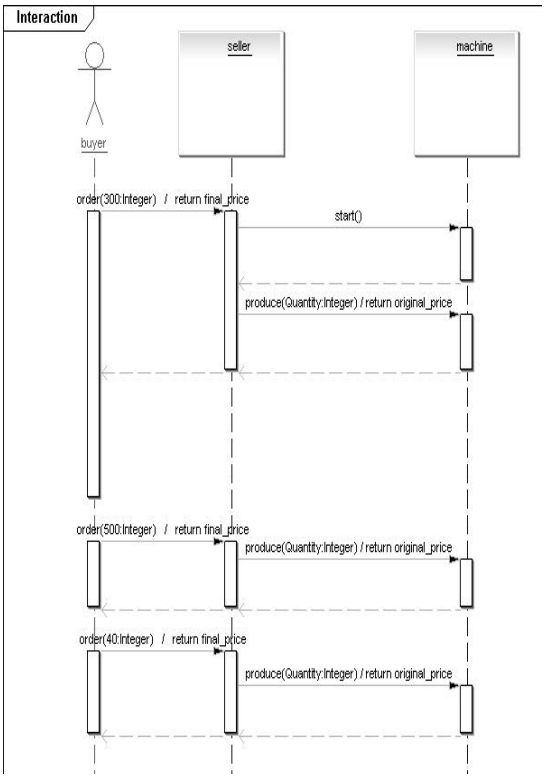

圖 3.購買流程循序圖

EventName(VariableName:VariableType) / return returnVariableName

EventName(VariableValue:VariableType) / return returnVariableName

這些訊息上的EventName會觸發目標參與物件的狀態機圖上的觸發事件，而 VariableName 及 VariableValue 會傳送給目標物件的狀態機圖做為參數，最後的 returnVariableName 則是將目標參與物件中的變數的值回傳給來源參與物件。另外要注意的是在循序圖中除了buyer屬於Actor所以不需要擁有狀態機圖，其餘的每個參與物件都必須要擁有自己的狀態機圖才能在本論文所提出的模擬器上正常運作。

本節剛開始的圖 3 描述了購買流程，而下面這張狀態機圖所描述的則是循序圖中的 seller 參與物件，這張狀態機圖含有一些狀態機圖中基本的表示方法，本論文所提出的模擬器可以正確的模擬這張狀態機圖所要表示的物件行為。

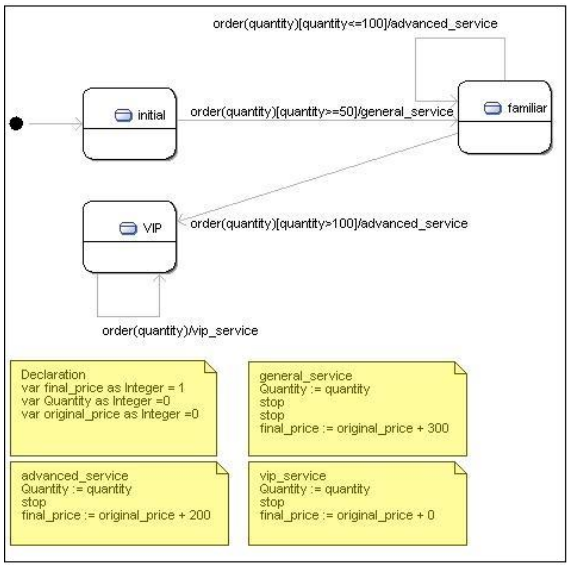

圖 4.seller 的狀態機圖

這張狀態機圖上描述 seller 擁有三種狀態：initial、familiar、VIP，想要從一個狀態轉移到另一個狀態必須要經過一個轉換動作(Transition)，而轉換動作的格式為觸發事件[成立條件]/活動(Event[Guard]/Activity)。initial 狀態跟 familiar 狀態間有一個轉換動作：order(quantity)[quantity>=50]/general_service，想要從 initial 狀態移到 familiar 狀態的話首先必須要發生 order 這個觸發事件，而 quantity 則會做為此觸發事件承接參數的變數，接著再對成立條件做檢查，如果成立條件也為真，那麼就會執行 general_service 這個活動，在本模擬器中 gerneral_service 這個活動的實際內容寫在 Note 上，因此就會存在一個以 general_service 為開頭的 Note。

在這張狀態機圖上有許多 Note 存在，Note 有分兩種，第一種是 Declaration Note，此種 Note 是用來宣告在這個狀態機圖中有哪些變數是可以用的，循序圖中拿來做為傳回訊息的變數也必須在這裡宣告，宣告的格式按照 ASML 的規定，而除了 Declaration Note 以外的 Note 都是拿來對應轉換動作的活動，撰寫的格式除了可以增加一些 stop point 外也是按照 ASML 的規定。

stop point 主要是為了決定在循序圖的活動長條中的某一段該做什麼事情，舉前面所舉的購買流程循序圖的例子來說，在循序圖中 seller 下的第一個活動長條該有兩個 stop point 去切割出三段 ASML 程式碼，第一個 stop point 會設在送出 start()訊息之前，第二個 stop point 設在送出 produce 訊息之前，有這兩個 stop point 就可以導出以下的執行流程

第一段 ASML 程式碼

stop(machine 正執行 start 觸發事件)

第二段 ASML 程式碼

stop(machine 正執行 produce 觸發事件)

第三段 ASML 程式碼

且因為此時 seller 因為剛初始化完所以處在 intial 狀態下，因此所執行的是 intial 狀態到 famliar 狀態的轉換動作上的活動，也就是 general_serivce，按照 stop point 的切割來執行的話就會變成以下的執行流程

第一段 ASML 程式碼(Quantity:=quantity)

stop(machine 正在執行 start 觸發事件)

第二段 ASML 程式碼(空白)

stop(machine 正在執行 produce 觸發事件)

第三段 ASML 程式(final_price:=original_price+300)

本模擬器在狀態機圖上可以支援轉換動作(transition)上的觸發事件(event)、成立條件(guard)、活動(activity)，不支援活動狀態(activity state)、結束狀態(final state)、內部活動(internal activity)、超狀態(superstate)、並行狀態。在循序圖上則可以支援同步訊息的傳遞、同步訊息的傳回及活性長條圖的邏輯，不支援自身呼叫、物件的產生與刪除動作、迴圈與條件式邏輯互動框、非同步訊息的傳遞及非同步訊息的傳回。

ASML 是 Abstract State Machine Language 的縮寫，這種語言在實現 abstract algorithm 時較為方便，如果需要 ASML 完整的文件的話可以從下載 Spec Explorer[3]並安裝，在安裝的資料夾裡就可以找到一些 ASML 的完整文件及範例程式碼，在此僅對 ASML 的一些特色做介紹。

ASML 是一種規格語言，規格語言和程式語言的差別在於規格語言著重在描寫結果是什麼，程式語言則著重在描寫如何執行出結果，因為規格語言

使用的指令都是很高階的，而這些高階的指令導致了效率較差。下面是用 ASML 來表示排序一串數字的邏輯的程式碼

```
var A = [3, 10, 5, 7, 1]
indexes = {0, 1, 2, 3, 4}
step until fixpoint
    choose i in indexes, j in indexes
            where i < j and A(i) > A(j)
        A(i) := A(j)
        A(j) := A(i)
```

以上這段程式碼所描述的語意是，在陣列中任選兩個數字並比較其大小，若較大的一個數字的 index 不是較大的話，那麼就交換這兩個數字的位置，這個任選兩個數字的行為持續到數字完成排序為止，由這個例子就可以看出相較於一般的程式語言來說，ASML 寫法非常簡潔。

Set

```
var s1 as Set of Integer  = { 5 , 4 , 3 , 2}
var s2 as Set of Integer  = { 5 , 3 , 4 , 2}
WriteLine(s1=s2)
```

這段 ASML 程式碼是在比較兩個 set 是否相同，以一般的程式語言(C、C++等)要達成一樣的功能可能需要寫一個兩層的 for 迴圈，但是在 ASML 上只是一行程式碼就可以完成了，另外 ASML 還提供了 Sequence、Map 等資料結構，使用方法與 Set 類似。

Data-oriented constraints

```
structure Rational
  A as Integer
  constraint NonZero : A <> 0
```

如果照上面這段程式碼的宣告方式的話，程式在執行的途中隨時都會幫程式開發者檢查程式執行的途中是否 A 的值曾經為 0，而這種隨時自動檢查物件內部的某些值是否符合某些規範應該是一個很有用的功能。

4 範例

我們先詳細說明圖 3 的範例。有一個購買流程的主要邏輯是，buyer 向 seller 下一筆訂單，seller 就會啟動 machine 然後再要求 machine 生產這批貨物，machine 則會告訴 seller 這批貨物的原始價格，seller 則告訴 buyer 這批貨物加上服務費後真正的價格是多少。

要求 machine 在第一次生產貨物前要先啟動的邏輯則畫在 machine 的狀態機圖中，如下圖。

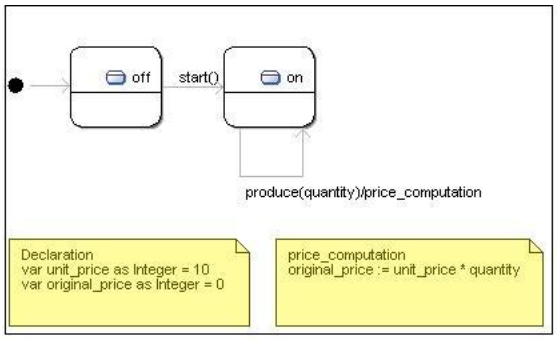

圖 5. machine 狀態機圖

從上圖中可以看到 off 狀態要先經過啟動才可以轉移到 on 狀態，在 on 狀態下才可以生產貨物並且計算這批貨物的價格，在圖上的 Declaration Note 中可以看到 unit_price 被設為 10，也就代表每單位的貨物價格為 10 元，而 original_price 則是用來記錄這批貨物原始的價格為多少，其公式為

原始價格=單價 x 數量

original_price=unit_price*quantity

除了主要邏輯外還有一些細節存在，seller 只有第一次向 machine 要求生產貨物前要先啟動機器。

另外就是 seller 在計算服務費時要按照客戶之前的購買歷史來決定，第一次下訂單的客戶要收的服務費是三百元，第二次的服務費是兩百元，第三次之後的訂單就要看第二次之後的訂單是否曾經有一筆大於一百單位的訂單，若有的話則不收服務費，若沒有的話則服務費仍然是兩百元，seller 的狀態機圖如圖 4 所示。

圖 4 的 Note 中的 final_price 記錄的是一筆訂單的原始價格加上服務費後的最終價格。圖上的

轉換動作上可以看到一些成立條件，例如[quantity>=50]代表對於未有交易記錄的客戶只接受大於50單位的訂單。

模擬器執行途中會詢問使用者 Actor 傳出的參數的值，以下面個例子來說，就是 300、500、40 這三個數字本來應該填為 VariableA、VariableB、VariableC 之類的名字，模擬器在執行的途中就會詢問使用者 VariableA、VariableB、VariableC 各自的數值為多少。

圖 3 中實際三次購買流程如下。第一次，buyer 跟 seller order 了 300 單位貨物，然後 seller 啟動machine,然後seller再要求machine製造300單位的貨物,接著machine就會將這批貨物的原始價格回傳給seller，seller收到這個價格後會再計算目前該跟buyer收多少服務費,將兩者相加後告知buyer，因為目前buyer是第一次來訂貨物，所以 buyer 該付的服務費是最高的 300 元，再加上所定的貨物的單價為10元，所以最後buyer該付的價錢是3300元。

第二次,接著buyer再定了500單位的貨物，seller要求machine製造500單位的貨物，接著machine 就會將這批貨物的原始價格回傳給seller，seller 收到這個價格後會再計算目前該跟 buyer 收多少服務費，將兩者相加後告知buyer，因為目前buyer是第二次來訂貨物，所以buyer該付的服務費是200元，再加上所定的貨物的單價為 10 元，所以最後 buyer 該付的價錢是5200元。

第三次，接著buyer再定了40單位的貨物，seller 要求 machine 製造 40 單位的貨物，接著machine 就會將這批貨物的原始價格回傳給seller，seller 收到這個價格後會再計算目前該跟 buyer 收多少服務費，將兩者相加後告知buyer，因為目前buyer是上次來訂貨物時定超過了一百單位，所以buyer已成為VIP，所以該付的服務費是0元,再加上所定的貨物的單價為10元，所以最後buyer該付的價錢是400元。

5 UML 模擬器

UML 模擬器由 UML 圖讀取器、循序圖模擬器、狀態機圖模擬器及 ASML 模擬器等四個元件所組成。圖 6 描述了 UML 模擬器的完整架構圖。

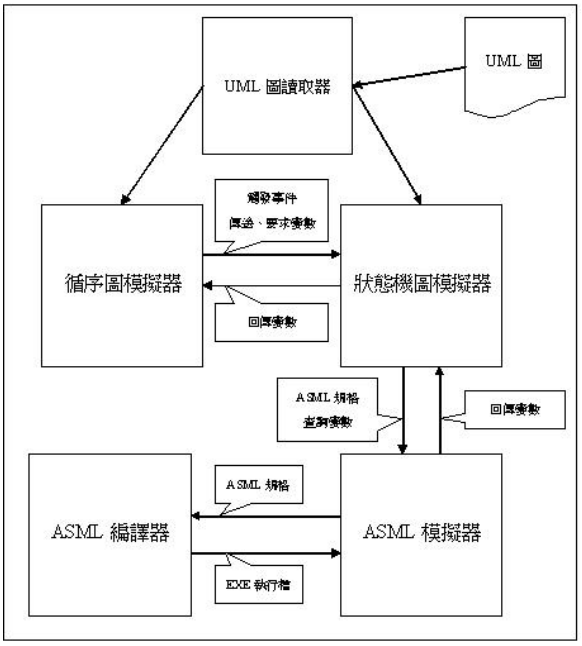

圖 6.UML 模擬器完整架構圖

UML 圖讀取器的主要功能是將存成 XML 格式的循序圖及狀態機圖讀取並解析，然後再轉成內部資料結構供循序圖模擬器及狀態機圖模擬器做後續的模擬。

循序圖模擬器負責模擬循序圖。每張循序圖描述軟體物件間的一種互動情境，明確描述物件間所有的訊息傳遞。循序圖模擬器將所有的訊息依發生的先後順序依序模擬。每個訊息的模擬可以分成三個步驟，第一個步驟先向訊息的來源物件的狀態機圖模擬器詢問訊息的參數值，第二個步驟將訊息的參數值傳遞給訊息的目標物件的狀態機圖模擬器並模擬訊息的執行，第三個步驟將訊息執行的結果再回傳給訊息的來源物件的狀態機圖模擬器。

狀態機圖模擬器模擬一個物件的行為。每張狀態機圖明確描述一個物件受訊息驅動所引發的狀態轉換。每一個狀態機圖模擬器都會記錄該物件目前所處的狀態，當循序圖模擬器詢問訊息的參數運算式的值時，狀態機圖模擬器即透過 ASML 模擬器來計算參數運算式的值，最後再將 ASML 模擬器

回傳的參數值回傳給循序圖模擬器。

當循序圖模擬器驅動訊息的執行時，狀態機圖模擬器即依據其所處的狀態、選擇符合訊息的邊來執行。一個邊的執行可以分成兩個步驟、第一個步驟先檢查成立條件是否滿足、如果不滿足、則不執行活動；否則、第二個步驟即執行活動。檢查成立條件是否滿足及執行活動這兩個步驟皆須透過 ASML 模擬器去計算。

ASML 模擬器內含一個 ASML 編譯器。每一個 ASML 模擬器也都有一個符號表(symbol table)記錄該物件的欄位(field)值。狀態機圖模擬器將訊息的參數運算式及狀態機圖上的成立條件及活動等 ASML 運算式傳送給 ASML 模擬器後，ASML 模擬器會先將其轉換成完整的 ASML 程式。接著 ASML 模擬器再透過 ASML 編譯器將其編譯成執行檔，然後 ASML 模擬器再執行此執行檔並截取它的輸出，最後 ASML 模擬器再根據輸出修改物件的欄位值。

以圖 4 的 general_service 活動為例，ASML 模擬器會將第二個 stop 之後的 ASML 運算式轉換成如下的 ASML 程式：

```
Main()
  let quantity = 300
  var final_price as  Integer  = 1
  var Quantity as  Integer  = 300
  var original_price as  Integer  = 3000
  step
    final_price := original_price + 300
  step
    WriteLine(final_price)
    WriteLine(Quantity)
    WriteLine(original_price)
```

6 結論與未來展望

本論文提出一個方法使得軟體開發團隊可以利用 UML 圖形及 ASML 規格去算出預期的輸出，並且可以在做軟體測試時拿著預期的輸出做為檢查所實做出的軟體是否有按照規格正確的實做，那麼就可以避免算出預期輸出時可能會牽涉到人工的問題，也因此可以做較大量的自動化軟體測試來確保軟體的品質。

本論文所提出的模擬器在做 ASML 的模擬時使用的編譯器速度並不理想，因此可能可以尋找更有效率的編譯器去做為 ASML 模擬的核心。

循序圖的模擬上可以增加自身呼叫、物件的產生與刪除動作、迴圈與條件式邏輯互動框、非同步訊息的傳遞等功能，而在狀態機圖的模擬則可以增加活動狀態、結束狀態、內部活動、超狀態、並行狀態等功能，如果這些功能都可以擴充上去的話，那麼這個模擬器就可以更貼近實際應用的狀態了。

參考文獻

[1.] Stephen J. Mellor and Marc J. Balcer. *Executable UML*. http://www.executableumlbook.com/

[2.] Terry Quatrani. *Introduction to the Unified Modeling Language*, 2006. ftp://ftp.software.ibm.com/software/rational/web/whitepapers/2003/intro_rdn.pdf

[3.] Spec Explorer. http://research.microsoft.com/specexplorer/

[4.] Mao Ye, Boqin Feng, Li Zhu, and Yao Lin. “Automated Test Oracle Based On Neural Networks.” In *Proceedings of 5th IEEE International Conference on Cognitive Informatics*, 2006, pp. 517-522.