一個以 UML 為輸入的即時嵌入式軟體之合成與驗證框架

A UML-based Synthesis and Verification Framework for Real-Time Embedded Software*

高新傑†、熊博安†、李宗演**、傅日明‡、施文彬§
†國立中正大學資訊工程學系嵌入式系統實驗室
**國立台北科技大學電子工程學系積體電路設計實驗室
‡正修科技大學電子工程學系
§經濟部漢航翔空工業股份有限公司國防系統與科技事業部工程處
E-mail: hpa@computer.org

摘 要

針對即時嵌入式系統之軟體開發需求我們設計了一個物件導向應用程式框架。我們採用目前廣泛使用的 UML 設計語言來建立系統規格模型。透過物件導向的設計來提高嵌入式系統的開發速度，可重複使用性及除錯能力。這個框架也整合了即時排程及正規驗證功能。驗證功能可以確保產生的嵌入式軟體符合系統規格。而排程功能可以確保產生的嵌入式軟體符合系統的即時特性，即滿足所有工作的時間限制。使用這個框架我們實作一個應用實例，即「門禁系統」。在實作過程中可以明確地看出使用這個框架來開發嵌入式軟體的優點。同時，也證明了框架的正確性及便利性。

一、簡介

在我們生活周遭，嵌入式系統佔了越來越重要的地位，小如行動電話，大如汽車，其中都包含了嵌入式系統。我們已經不知不覺中逐漸習慣依賴這些設備，因此嵌入式系統的設計成為很重要的課題。但是嵌入式系統設計的特性在於它的行為很複雜卻又擁有非常緊迫的上市時間限制。傳統設計嵌入式系統必須仰賴工程師的經驗。因此如果是面臨一個新的平台，往往必須重新學習新的軟硬體。而程式完成後的除錯更是困難，傳統使用測試為基礎的除錯技術非常花費時間卻不能保證整個程式經過測試流程以後就一定不會出錯。

由於以上的限制造成嵌入式系統設計時的許多困難。為了解決這些問題，我們提出了一個物件導向嵌入式軟體開發框架。在其中整合了 UML 塑模技術，物件導向技術，軟體排程技術，軟體驗證技術及自動化程式碼生成技術。

UML [9] 是目前工業界最廣泛使用的塑模語言之一，讓設計師以 UML 為輸入的模形，可以減少設計師建立模型或學習新語言的時間，並且利用 UML 的特性完整的描述系統的行為。物件導向技術使得軟體元件能夠擁有重複使用性，設計師不需要花很多的時間去適應新的平台，他可以直接使用現存的軟體元件加以整合，大大地減低了開發的時間。軟體排程技術確保產生的程式能夠符合使用者指定的時間限制，對於即時嵌入式系統的設計有很大的幫助。軟體驗證技術可以讓設計者經由正規驗證的方式，完整地驗證系統行為的正確性，避免系統執行可能發生的錯誤。自動程式碼生成技術讓使用者可以減少自行撰寫程式碼的需要，可以減少人為的程式錯誤及撰寫程式碼的時間。

接下來的幾節，針對我們提出的系統分別說明。第二節介紹一些相關的研究，第三節中介紹系統的完整流程，第四節中介紹一個以此系統實作的例子－門禁系統，在這個例子中我們可以詳細的了解使用這個系統開發嵌入式軟體的優點。第五節則是結論。

二、相關研究

目前有許多自動產生嵌入式系統程式的研究，例如美國軍方 MoBIES (Model-Based Integration of Embedded System) 計劃中的 AIRES [1] 工具及 CMU 提出的 Time Weaver [2] 工具。這兩個工具都提出使用物件導向框架來輔助發展嵌入式系統的概念。

MoBIES 中包含了許多個子計劃都是相關於嵌入式系統軟體自動產生技術，其中的 AIRES 著眼點在於嵌入式軟體的建置模型，它將嵌入式軟體的產生分成許多步驟，分別使用不同的描述模型來建立系統。包括了行為模型，軟體結構模型，執行

* 本論文為行政院國家科學委員會專題研究計畫之部分研究成果 NSC-91-2213-E-194-008 (即時嵌入式軟體之合成工具設計(1/3))

模型及生成碼模型。利用這些模型可以表示系統的行為及元件的關係，並且提供效能分析及自動產生程式碼的依據。他們提出框架中的軟體元件應該以meta-model的方式存在，提供相關於軟體行為，結構，環境及效能的資料。

Time Weaver 著重的研究在於提供設計時使用的軟體模型框架，建立驗證模型及可執行的程式碼，它針對嵌入式軟體中可能出現的一些非功能性規格 (例如時間點，服務品質及可靠度) 提供分析的機制，確保這些規格能夠被滿足。它透過元件模型的結合來產生程式碼。並且使用 coupler 來表示元件之間的關係，不同的 coupler 可能表示不同的關係，包括了時間，資料流等。最後使用語意維度 (semantic dimensions) 來提供針對相同物件的不同觀點。

這些工具都提出了使用元件模型產生嵌入式系統軟體的方法，但是它們都使用自己制定的模型，因此使用者必須另外學習它們使用的塑模語言，造成使用上的困難。另外它們必須使用額外的時間評估工具來判斷程式執行時是否會不滿足時間限制，對於軟體功能的正確性也沒有辦法提供使用者可靠的保證。即時特性及正確性都是嵌入式系統很重要的元素，不能確保這些特性可能造成系統的不穩定，甚至造成很大的損失。因此我們有必要確保這方面的正確性。

我們的框架中使用 UML [9] 作為系統的輸入模型，由於 UML 是工業界最廣泛使用的塑模語言之一，因此大部分的使用者可以很快的學會使用這套工具。另外我們整合了系統排程及正規驗證的方法來確保系統的即時特性及功能正確性。透過系統排程，產生的程式可以滿足使用者限定的時間限制，保證系統執行時能夠符合時間限制完成。正規驗證可以避免傳統使用測試方法可能有的涵蓋率問題，它可以保證系統完整滿足規格的屬性。

三、系統架構

這個系統架構的完整流程可見圖 1。圖 1 中實線代表的是工作流程，當一個工作完成接下來就進入箭頭指向的工作繼續執行。虛線代表的是資料流程，被虛線箭頭指向的工作需要虛線來源的這筆資料來完成工作。

整個系統的流程可以分成兩大部分。第一部份提供前端模型產生及驗證排程工作；第二部分提供後端程式碼生成工作。前端又分成四個部分，分別是 UML 模型輸入部分，程序排程部分，系統驗證部分及排程器生成部分。後端則分成兩個部分，分別是可重用元件對應部分及程式碼生成部分。

以下就針對各部分功能進行詳細的解說：

- UML 模型輸入

  UML (Unified Modeling Language) [9] 即為統一塑模語言，是一個在軟體工業界廣泛使用的塑模語言。其制定了標準化的概念及圖示來塑造系統模型。我們選用 UML 中的三種圖作為系統的輸入模型，這三種分別是類別圖 (Class Diagram)，順序圖 (Sequence Diagram) 及狀態圖 (Statechart)。其中類別圖可以用來表示軟體中的物件類別，各個物件之間相互的關聯及繼承關係。順序圖用來表示物件之間的訊息傳遞，利用順序圖我們可以清楚的了解系統執行的順序及物件間訊息傳遞的行為。狀態圖表示物件內部的行為模式，透過狀態圖中狀態的轉換，我們可以了解一個物件執行時

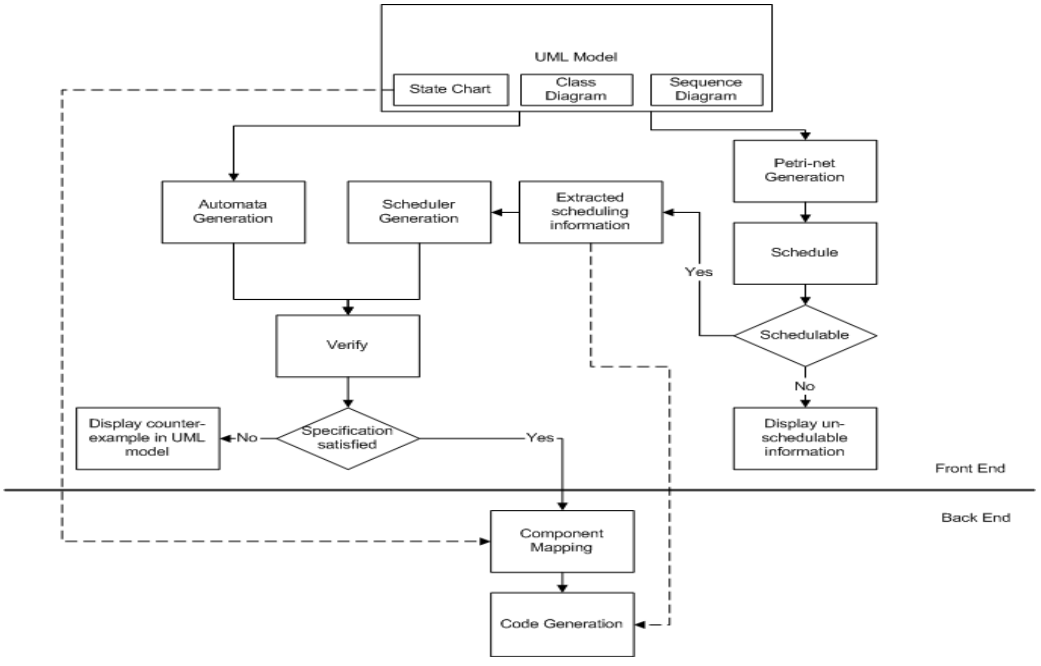


圖 1 系統完整流程

發生的行為。

我們針對即時系統的需求對這三種圖進行了小幅的修改，以描述更多即時嵌入式系統相關的行為。

在類別圖中，我們增加了硬體及軟體物件的分別。在圖中我們使用灰色為底色的類別方塊作為硬體物件，使用白色為底色的類別方塊為軟體物件。另外使用虛線作為軟硬體物件之間的關聯，實線作為軟體物件之間的關聯。另外使用者可以在 Operator 欄位中定義的函式加上 period 或 deadline 關鍵字，用來指定某個函式為一個時間驅動函式，這個物件可能以一個週期自動的執行或是到達期限後自動執行。

在順序圖中，我們增加了一個稱為狀態標籤 (state marker) 的符號 [3] ，其中的狀態名稱是相對於狀態圖中的狀態名稱，它可以讓狀態之間的訊息傳送順序更加明確。

在狀態圖中，我們針對時間驅動函式增加了三個關鍵字，分別是 start，stop 和 reset。它們分別可以啟動，結束及重設一個時間驅動函式。除了這三個關鍵字以外，我們另外還增加一個關鍵字 time-out，它可以用來指定函式的即時特性。

- 程序排程

嵌入式系統通常亦為即時系統，許多嵌入式系統甚至需要特別強調程式即時的特性。例如一個病患監控系統，這個系統相關於病患的身體資訊，如果不能即時判斷這些資料並且產生立即的反應，可能就會對病患的生命安全造成影響。因此當設計一個嵌入式系統時需要考慮到這個系統是否有任何的時間限制，這些時間限制可能是區域性或全域性的，它們的分別在於這個時間限制相關於整個系統的位置。當產生程式時，我們必須確保這些限制可以確實的被達成，避免實際執行時可能超出時間限制造成的影響。因此我們在產生程式的時候必須使用適當的排程方法達成這樣的目的。

我們採用的排程方法為「延伸半靜態排程」(Extended Quasi-Static Scheduling, EQSS) [4]及「類似動態排程」(Quasi-Dynamic Scheduling, QDS) [5] 排程方法。這二個方法針對系統中指定的全域時間限制及區域時間限制將程序進行排程，在排程後調整過先後順序的程序，讓系統可以依序進行而不會牴觸系統的各種時間限制，用以達到即時系統的需求。

為了使用這個排程方法，我們必須先要將一開始的 UML 模型轉換成它的輸入模型。它使用的輸入模型是 Petri-net。Petri-net 是在學術界廣泛使用的一種塑模語言，它的特點是可以明確的表現程式中並行的行為，因此大量的被使用在嵌入式系統行為分析之上。Petri-net 中包含了兩種節點，分別是轉換 (transition) 節點及空間 (place) 節點。為了產生整個系統的 Petri-net 模型，我們使用順序圖來轉換產生 Petri-net 模型，轉換的方法如下：

1. 將順序圖中的每個訊息轉換成一個轉換節點，它的進入箭頭，輸出箭頭，輸出的空間節點。如果是一個初始訊息則不會產生它的進入箭頭 圖 2(a)為一個初始訊息產生的 Petri-net，圖 2(b)是一般的訊息產生的 Petri-net。

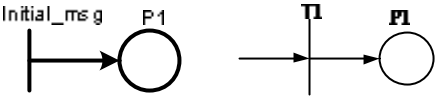


圖 2 (a)　　圖 2 (b)

2. 如果一個訊息有包含執行條件的話，在轉換節點的進入箭頭上標上這個執行條件。如圖 3，我們將判斷條件加在箭頭上，來確保符合判斷條件時進入轉換節點。

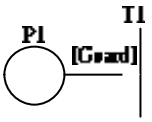


圖 3

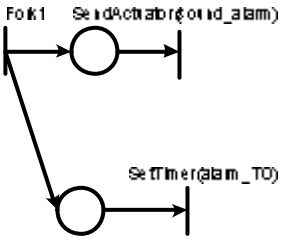


圖 4

4. 如果訊息是以迴圈型態傳送，將迴圈中一系列的訊息傳遞按照以上方法產生相對應的 Petri-net 節點，然後在最後一個轉換節點後加上一個輸出箭頭指向進入此迴圈的分支 (branch) 空間節點。這個分支空間節點指向迴圈的輸出箭頭上需

要加上進入此迴圈的條件，另外一個輸出箭頭則指向離開迴圈後進入的轉換節點。圖 5 為一個迴圈型態 Petri-net 的例子，最左側的空間節點為一個分支節點，它的輸出箭頭上有一個判斷是否進入迴圈的條件。最右邊的轉換節點有一個箭頭指向它，表示了迴圈中回到原點執行的行為。當不滿足進入迴圈的條件就會透過往下的箭頭進入 ClearDisplay 的轉換節點中。中間的空間節點表示了輸入時間結束的行為，如果輸入時間已經結束了，就會在這個節點離開迴圈，進入 TimeOut 這個轉換節點。

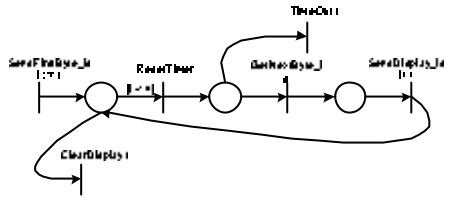


圖 5

5. 不同順序圖產生的獨立的 Petri-net，如果一個的開頭轉換節點和另一個的結尾轉換節點相同，我們可以直接用單一轉換節點表示，並且把兩個 Petri-net 連結在一起。在圖 6 (a)中，有兩個獨立的 Petri-net，由於它們分別擁有的開頭轉換節點及結尾轉換節點 T1，因此我們可以將兩個圖連接在一起，結果如圖 6 (b)。

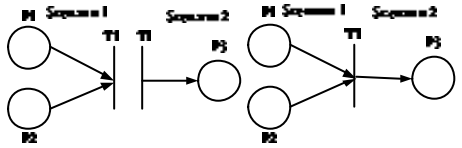


圖 6 (a)　　圖 6 (b)

6. 有先後關係的不同的順序圖產生的 Petri-net 我們可以使用一個分支空間節點把它們連結在一起。圖 7 中，虛線方框中表示個別的 Petri-net，我們使用分支節點 P2，將這些 Petri-net 連起來。

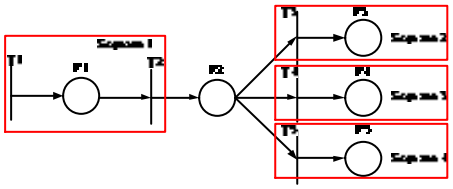


圖 7

7. 每一個順序圖產生的 Petri-net 都有一個結尾轉換節點。例如圖 8 中最右邊有一個節點 End2，當左邊的 Petri-net 結束，就指向這個節點 End2

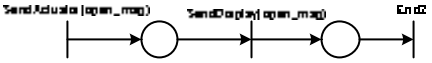


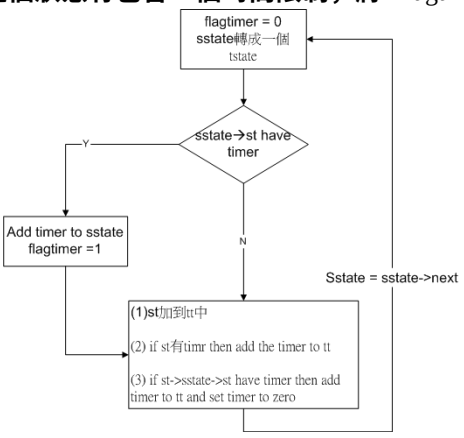
圖 8

透過以上的幾個規則，我們可以自動的將順序圖轉為 Petri-net 並且完整保留時間的限制資訊。

- 系統驗證

在系統驗證部分我們採用的是正規驗證的方法，所謂的正規驗證就是使用數學的方法去證明整個系統在驗證的條件下是一定正確的。傳統的驗證方法是使用模擬測試的方法，由於嵌入式系統的功能日漸複雜，傳統的模擬測試就會發生瑕疵。因為功能龐大，測試的範圍也相對的擴大，一方面造成測試時間拉長，減低產品上市的速度，另一方面造成測試輸入資料設計困難，很多可能的問題也許沒有辦法被發現。使用正規驗證方法就可以避免這樣的問題，經由正規驗證後，可以確保系統完全正確，不會保留沒有測試到的錯誤。

在這邊我們使用的正規驗證工具是 SGM (State Graph Manipulators) [6] [7] 。SGM 使用延伸時間自動機 (Extended Timed Automata) [10] 做為輸入模型，因此我們必須先將輸入中的狀態圖改成時間自動機。圖 9 是將狀態圖轉成時間自動機的方法，其中 sstate 表示狀態圖中的狀態，st 表示狀態圖中的轉換箭頭，tstate 表示時間自動機中的狀態，tt 表示時間自動機中的轉換箭頭。在第一個程序中，狀態圖狀態被轉為一個相對應的時間自動機狀態。接下來的判斷中，如果這個狀態有包含一個時間限制，將 flagtimer



設為 1，如果沒有就保持原值為 0，接著將狀態圖中的轉換箭頭改成時間自動機的轉換箭頭，如果轉換箭頭上有時間限制就將這個時

圖 9 時間自動機生成演算法

間限制加到產生的轉換箭頭上，如果在一組程序上包含了時間限制，也要將時間限制加到轉換箭頭之上。經由這樣的程序就可以自動產生相對應於原始系統的一組時間自動機提供驗證工具使用。

- 排程器生成

經由前述的排程步驟之後，我們可以得到一組排程後的程序執行順序，為了控制產生的程式能夠依照這個排程順序執行，我們必須建立一個控制器。這個控制器可以控制各個物件內狀態圖變化，阻止狀態圖變化可能超出設定的時間限制。我們利用前述排程步驟產生的排程資訊建立一個時間自動機。這個時間自動機和其他的物件傳送訊息以了解其他的物件內部狀態的改變，另外它也依照排程資訊傳送訊息控制其他物件狀態的改變。

- 可重用元件對應

由於嵌入式系統可能包含了許多不同的硬體或作業系統，當嵌入式系統程式設計師必須針對新的平台開發應用程式時，他需要先建立相關於這個硬體平台及作業系統的軟體元件。在建立這些軟體元件之前，他還要花費大量的時間學習軟硬體的相關資料，這對於開發時間緊迫的嵌入式系統開發流程來說是不被允許的。如果可以將這些相關於硬體平台及作業系統的軟體元件蒐集起來，整理於資料庫中，就可以在需要的時候立刻拿來使用，避免再學習軟硬體相關資料的時間。

這些軟體元件我們必須設計一組固定的介面，因此就算使用不同的硬體，設計師也知道要利用哪些函式來呼叫這些硬體的功能。我們針對一般硬體會有的行為，將最基本的介面定為四個，分別是：Init(), reset(), write(), read()。每個硬體相關的軟體元件都應該包括這些基本介面。當使用者自行撰寫硬體相關軟體元件時也必須實作這些介面，提供以後的使用者使用。

因此這部分要進行的工作是將類別圖中定義的硬體物件及軟體物件由平台資料庫中取出相對應的可重用元件。例如在類別圖中定義了一個 LCD 硬體物件，在生成應用程式時同時也必須產生能夠控制 LCD 的驅動程式。在我們的框架中定義了一組硬體的驅動程式或應用程式介面，VERTAF 透過使用者的平台定義選取適合的可重用元件，再將這些軟體元件與其他的使用者自訂的部分作結合。

- 程式碼生成

這部分要進行的工作是將以上的程式生成可執行的程式碼。我們採用的方法是由 Miro Samek 提出的量子程式法 (Quantum Programming) [8] 。這種做法是將每個物件中的狀態圖都當作獨立的單元，每個單元都自行運作，單元間只藉由訊息的傳遞來建立關聯。使用這種做法，我們可以直接將使用者輸入的狀態圖轉成相對應的程式，我們產生的程式使用圖 10 的架構。

圖 10 中最下層的是硬體部分，我們目前支援的硬體平台是使用 StrongArm 處理器的發展平台。硬體層之上是我們支援的兩種作業系統，一個是 Embedded Linux，另外一個是 MicroC/OS。Embedded Linux是將 Linux 精簡後產生的一個作業系統，由於體積減小適合儲存空間較小的嵌入式系統使用。MicroC/OS 是一個專門為嵌入式即時系統設計的作業系統，它使用 MicroHAL 作為與硬體溝通的介面，因此可以很容易的轉移到別的硬體平台之上。再上一層是 Quantum Framework，我們利用 Quantum Framework 中的 ActiveObject 類別來產生相對於類別圖中的各個物件，每個物件自行依照其內部定義的狀態圖運作。我們產生的程式分為兩部分，一部分是由使用者輸入的 UML 模型產生的主程式，另一部分是由排程器狀態圖產生的控制器程式。由 UML 模型產生的程式利用 Quantum Framework 提供的 ActiveObject 類別產生個別的物件，排程器狀態圖同樣利用 ActiveObject 類別產生另一個物件，這個物件可以透過訊息的傳遞，控制主程式其他物件內部狀態圖的變化，以此達成控制的目的。

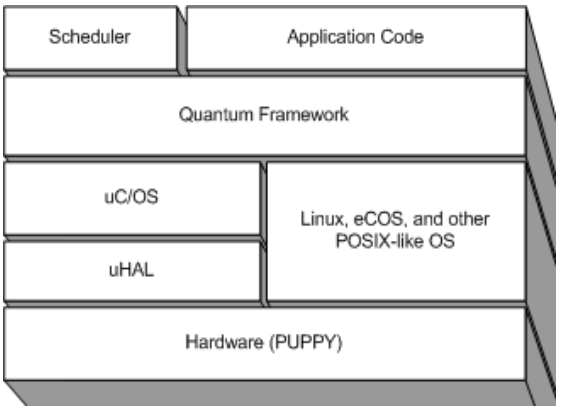


圖 10 產生程式碼架構

三、應用實例

我們使用一個實例來實驗這個框架的運作。我們採用的例子是一個門禁系統，這是一個可以控制大門開關的系統，使用者可以透過鍵盤輸入帳

號，也使用麥克風輸入一組語音密碼，這個系統利用硬體的語音辨識晶片判斷使用者是否為同一人，是否說出正確的關鍵字。只要使用者輸入正確的帳號密碼就可以開啟大門，反之如果使用者連續輸入錯誤三次，就會有警鈴聲響起。這個系統包括了六個硬體元件及六個軟體元件，圖 11 為這個系統的類別圖，其中方塊為各物件，灰色方塊即為硬體物件，白色方塊為軟體物件。實線表示各軟體物件之間的關係，虛線則表示硬體物件及軟體物件之間的關係。線上的文字表示這條連線代表的關係。

軟體物件分別是 Input，Checker，Controller，Actuator，DBMS 及 Display。Input 物件處理使用者的輸入，包括使用者利用鍵盤輸入帳號及使用音效裝置輸入語音，因此 Input 物件分別與 Keypad 物件及 Audio 物件建立一個關聯可以存取這些硬體資源。Checker 的工作是判斷使用者輸入的帳號和語音是否符合，因此 Checker 與 Input，DBMS 及 VoiceRecognizedIC 這三個物件建立關聯。Checker 經由與 Input 的關聯可以取得使用者輸入的帳號及語音資料，另外透過 DBMS 的功能取得資料庫中的語音資料，VoiceRecognizedIC 物件則提供硬體資源協助比對聲音資料。Controller 物件為判斷及控制行為的主要物件，它透過與 Checker 的聯結取得帳號判斷的結果，透過與 Display 物件的聯結可以把目前狀態訊息顯示在螢幕上，透過與 Actuator 的聯結，可以將判斷後下的指令交給各硬體去執行。Actuator 專門負責控制各硬體，透過與 Controller 的關聯，它可以取得指令，再依照指令去控制硬體。例如它利用 Audio 物件播放警鈴聲，利用 LED 物件表示門的開啟與否。DBMS 物件提供存取資料庫的功能，它與 Checker 及 FlashROM 之間的聯結讓它可以從 Checker 中得到檢索的指令，並且從 FlashROM 中取得符合檢索指令的資料。Display 物件提供顯示畫面的功能，它與 Input 及 Controller 物件之間的關聯表示它從這兩個物件得到顯示指令的關係，與 LCD 物件的連線則讓它實際訊息顯示在 LCD 上。

順序圖包含了五個圖，每一個表示一組可能發生的執行順序。限於版面，我們只詳細的解釋其中一個圖形，圖 12 這個順序圖表示可以物件之間訊號的傳遞，從使用者輸入帳號開始一直到液晶螢幕上顯示開門成功的訊息才結束。圖中每一欄為各個物件的執行時間。最左欄是使用者，當使用者按下按鈕就會發生系統的起始事件，傳送 PushButton 這個訊息給 Input 物件，緊接著 Input 物件會把這個收到的鍵傳送給 Display 物件讓他把使用者輸入的帳號顯示在螢幕上。接著 Input 物件會傳送訊息給 Timer 物件啟動一個計時器，這個計時器的目的是為了建立一個使用者輸入的時間限制，避免使用者輸入帳號的第一個字元後沒有繼續輸入完成的情況。計時器執行的同時，Input 物件也繼續接受使用者輸入的帳號訊息，一直接收到四個位元組以

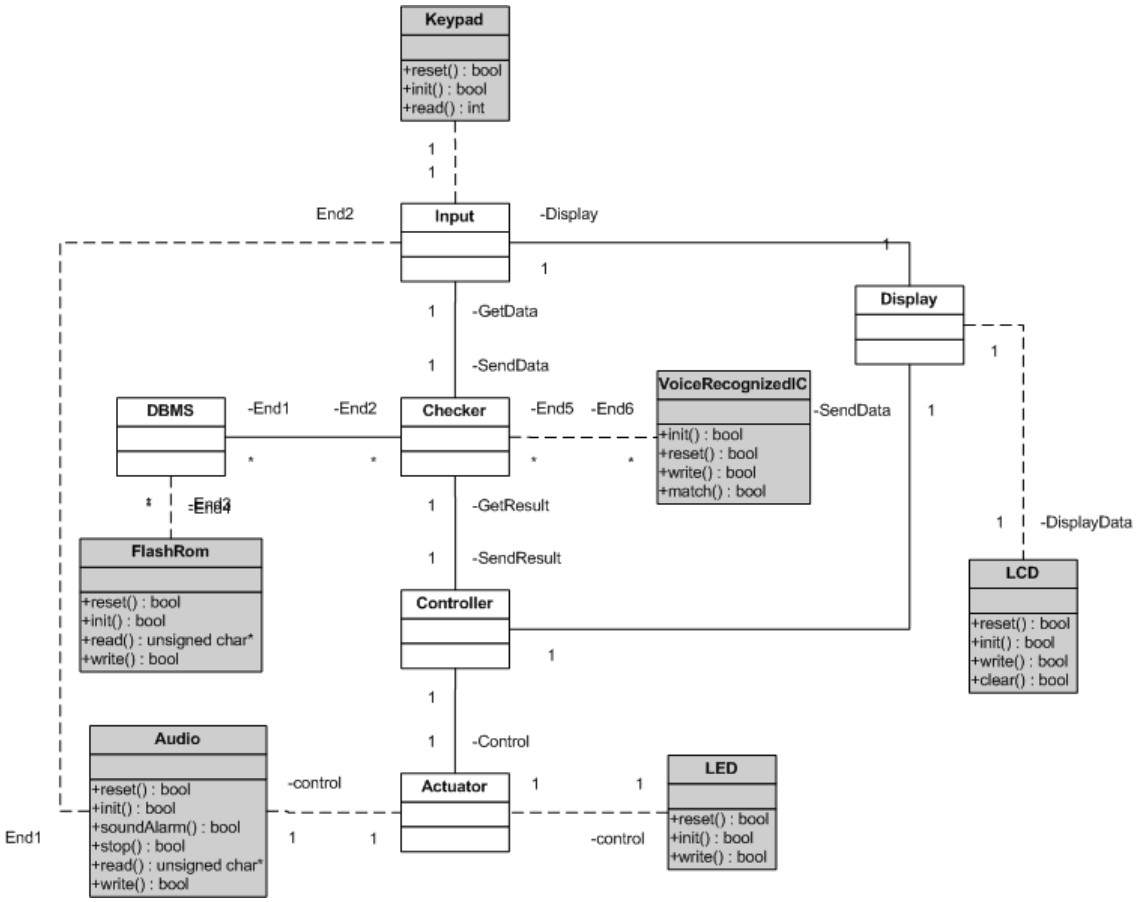


圖 11 門禁系統類別圖

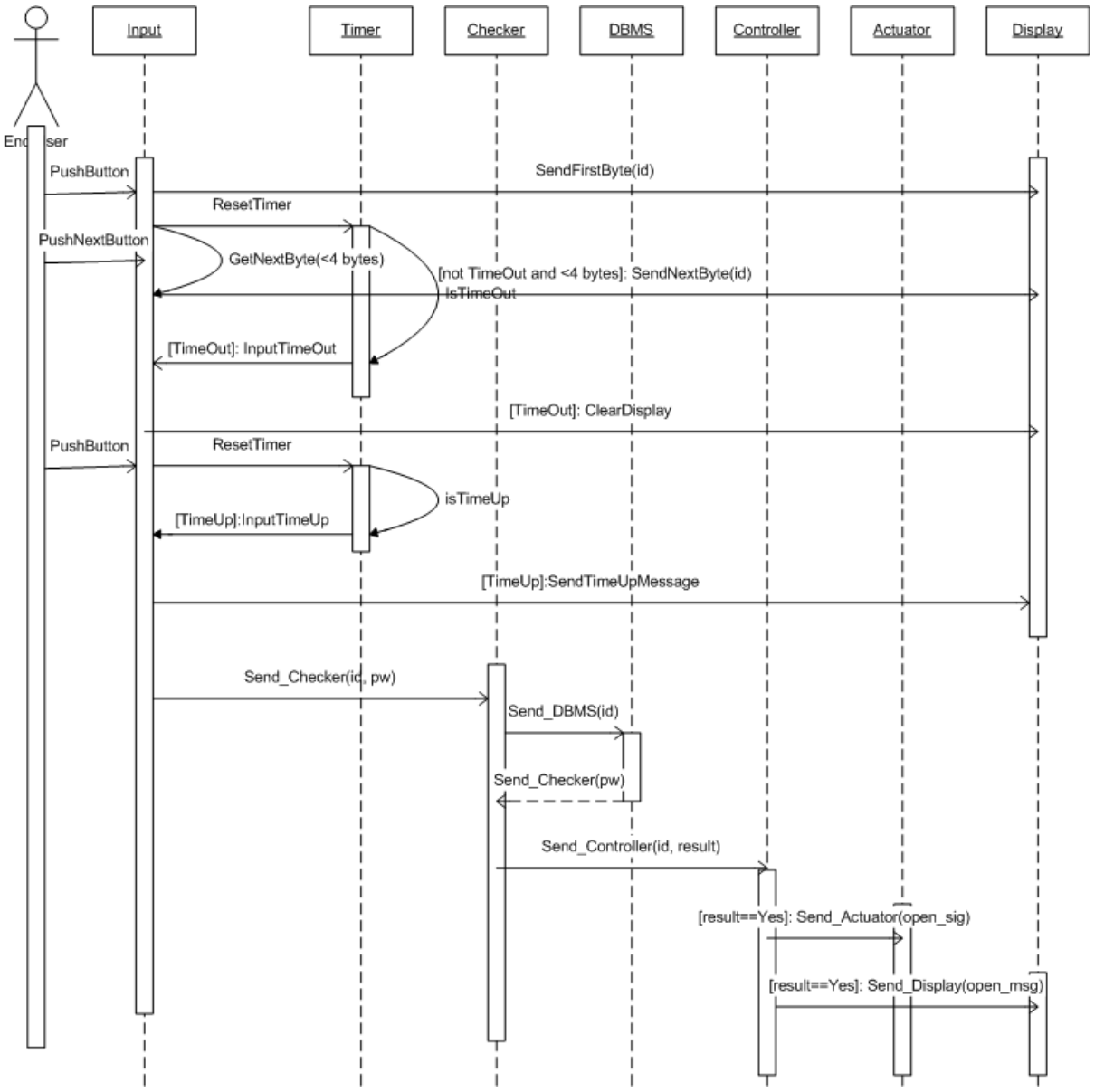


圖 12 門禁系統順序圖之一

後才會結束數入帳號的動作。

輸入帳號的動作結束後如果使用者再送出按下按鈕的訊息就會啟動錄音的行為，同樣的也是啟動一個計時器計算錄音的時間，時間到了就送一個訊息回 Input 物件終止 Input 上的錄音動作。

接下來 Input 物件會將剛剛取得的帳號和語音密碼利用 Send_Checker 訊息傳送給 Check 物件，Checker 物件接著傳送 Send_DBMS 訊息給 DBMS 以取得資料庫中相對應於使用者帳號的語音密碼資料。接著 DBMS 將密碼回傳給 Checker 讓它判斷帳號和密碼是否符合。然後 Checker 物件將比對的結果和使用者帳號傳送給 Controller 物件，Controller 物件則判斷這個結果，如果結果是正確的，Controller 分別傳送訊息給 Actuator 物件及 Display 物件讓它們執行開門及顯示開門訊息的動作。Checker 傳送給 Controller 的帳號資訊則用於檢查同一個帳號是否出現連續輸入錯誤的情況，如果超過三次就啟動警鈴。

圖 13 為 Input 物件的狀態圖，由 Init 狀態開始進行初始化的工作，接下來進入 Idle 的狀態，當有訊息傳送到的事件發生就進入 Read_ID 狀態。在

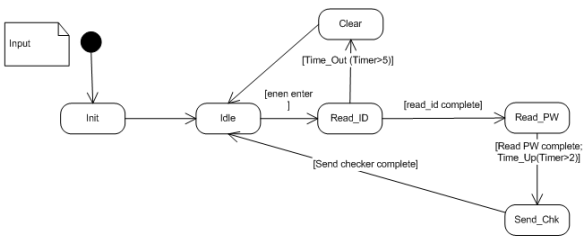


圖 13 門禁系統輸入物件狀態圖

這個狀態中，Input 物件讀入四個位元組的帳號。

如果時間超過五分鐘就進入 Clear 狀態將螢幕上的訊息清除，並且回到 Idle 狀態等待接下來的輸入。如果五秒內輸入完成就會進入 Read_PW 狀態。在這個狀態中開始錄音，2 秒後進入下一個狀態 Send_Chk。Send_Chk 狀態將帳號及密碼傳送到 Checker 物件。

UML 模型完成後經由我們的工具可以產生相對應的時間自動機及 Petri-net，利用這兩種圖可以產生系統驗證及排程結果。如果驗證結果不正確會產稱相對應於 UML 圖上的反例，提供使用者修該的參考，使用者修改完 UML 模型後可以再從新驗證，一直到可以完整的通過驗證才產生最終的程式碼。

四、結論

在這篇論文中，我們提出了利用物件導向模型來建立嵌入式應用軟體的方法。首先我們使用工業界廣泛使用的 UML 作為系統輸入模型，提供建立模型的便利性。再來我們使用驗證及排程的技術，一方面確保程式功能的正確性，另一方面確保程式能夠滿足即時系統的時間需求。接著我們使用量子程式技術產生各元件獨立運作的嵌入式系統程式。另外嵌入式軟體元件資料庫的使用讓使用者可以直接取用現存的嵌入式系統元件，免除重新開發常用元件及熟悉軟硬體平台的時間。

最後，我們使用一個門禁系統的例子來證明我們框架的正確性，透過例子的實作也可以發現使用這種方法設計嵌入式軟體的優點。

參考文獻

[1] S. Wang, S. Kodase, and K. G. Shin, 2002, Automating embedded software construction and analysis with design models, Proceedings of International Conference of Euro-uRapid 2002, Frankfurt, Germany, December 2002.

[2] D. de Niz and R. Rajkumar, 2003, Time Weaver: A software-through-models framework for embedded real-time systems, Proceedings of the International Workshop on Languages, Compilers, and Tools for Embedded Systems, ACM Press.

[3] B. P. Douglass, 1999, Real-Time UML 2nd Edition, Addison-Wesley.

[4] P. A. Hsiung, F. S. Su, 2003, Synthesis of Real-Time Embedded Software by Timed Quasi-Static Scheduling, Proc. of the 16th International Conference on VLSI Design, (VLSI'2003, New Delhi, India), pp. 579-584, IEEE CS Press, January 2003.

[5] P. A. Hsiung and C. Y. Lin, 2003, Synthesis of Real-Time Embedded Software with Local and Global Deadlines, Proc. of the IEEE/ACM International Symposium on Hardware-Software Codesign and System Synthesis (CODES-ISSS 2003), ACM Press, California, USA, October 2003.

[6] P. A. Hsiung and F. Wang, 1999, User-friendly Verification, in Proceedings of IFIP TC6/WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols & Protocol Specification, Testing and Verification, (FORTE/PSTV '99), Octorber 1999.

[7] F. Wang and P. A. Hsiung, Efficient and User-Friendly Verification, IEEE Transactions on Computers, Vol, 51, No. 1, pp.61-83, January 2002.

[8] M. Samek, 2002, Practical Statecharts in C/C++ Quantum Programming for Embedded Systems, CMP Books.

[9] J. Rumbaugh, G. Booch, and I. Jacobson, 1999, The UML Reference Guide, Addison Wesley Longman.

[10] R. Alur and D. Dill, 1994, Automata for modeling real-time systems, Theoretical Computer Science, Vol. 126, No. 2, pp. 183-236, April 1994.