解決 SDL 與 UML 在即時軟體設計上的問題

作者: Emmanuel Gaudin, PragmaDev　　譯者: 林舜英, 皮托科技股份有限公司

在即時軟體設計上，SDL與UML的使用分別面臨各種不同的難題。SDL-RT得利於C與SDL的完整性，另一方面又具備UML易於了解的視覺化。因此，一個以SDL-RT為基礎的工具，便可以解決SDL與UML兩者不能同時兼顧的重重限制。

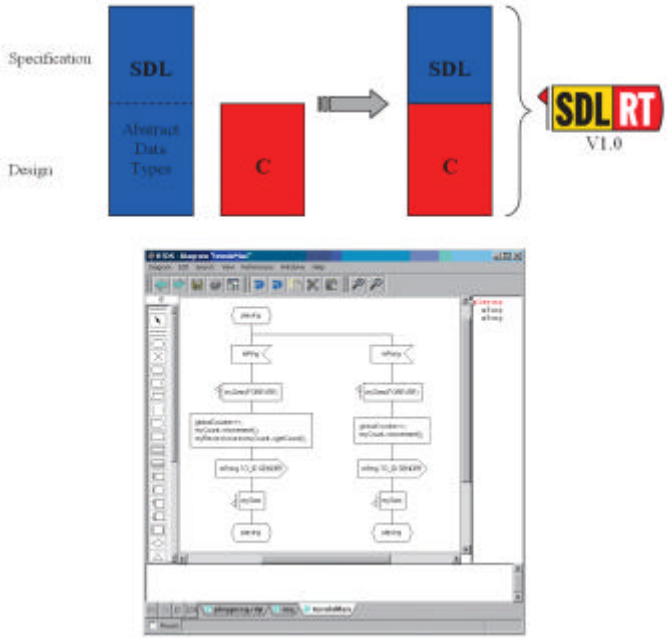

圖1. SDL-RT針對訊息的送收，提供圖形化的表現方式，如信號（semaphore）的取得與釋放。

SDL（規格描述語言，Specification and Description Language）為國際電信聯盟（ITU）為了描述電信協定所制定。實驗結果顯示，SDL的基本原理能夠擴展到許多各方面領域中，主要的原因是其圖形化的顯示方式，能夠清楚描述系統的行為與架構。但當涉及執行與整合到Target上時，便會因為失去對目標程式碼的控制而使得SDL的資料型態與其高階概念變成大麻煩。此外，某些即時概念如信號（semaphore）與指標（pointer）並未包含在SDL語言中，因此，SDL發展者開始在SDL的圖表中寫入C，以填補此部分的缺點。又因當時市面上並未有任何支援SDL與C混合寫成的工具，因此使用者必須自行撰寫 C code generator，以從SDL與C混合使用的圖表中匯出程式碼。

由上述所提到的問題看來，我們不難發現SDL開始被廣泛地使用。這也就是為什麼包含UML特點的SDL-RT能夠解決SDL語言與生俱來的缺陷。

即時與嵌入式軟體以即時作業系統或排程為基礎，基本構成元素為任務（task）。同時執行好幾個任務（task）便可完成一個基本的功能用途（function）；好幾個功用同時運作便可以完成更複雜的功用（function）；直到最後完成整個應用（application）。也就是說，SDL的架構能夠在功能方塊中聚集任務（task），接著在更高階的方塊中聚集所要執行的功用（function），一層包一層，直到完成整個系統。這樣的圖形化顯示架構能夠完美詮釋任何即時（real-time）系統，亦可完整描述系統中不同功能間的溝通介面。介面藉由交換訊息格式所定義，並依據構成資料與劇本來描述一連串的交換訊息。SDL信號能夠附帶輸入參數，這些參數以抽象資料型態（ADT）為基礎，可包含靜態介面的完整描述。訊息序列圖表（Message sequence charts, MSC）也是ITU的標準，其針對動態觀點，像是系統內部或外部模組（如驅動器）間的資訊交換，提供圖形化的顯示介面。即時系統仰賴眾多同時執行的獨立任務，因此當任務閒置時，如何不浪費CPU的時間是非常重要的一點。基於此，大部分的即時應用程式都是根據有限狀態機（Finite state machine, FSM）的原理而設計，只要一完成自己的工作時，便開始等候下一個RTOS物件（如訊息佇列）。

SDL的有限狀態機很適合透過圖表的方式來描述這種系統行為。從'92版開始，SDL在所有圖形化階層皆使用物件導向，所以建立軟體元件專門的函式庫是確實可行的。

理論上，SDL似乎是針對即時系統規格與設計的最佳語言，但技術上的執行卻相當困難。當涉及實務設計，因為那些操作ADT的語法是被定義用以制定協定規格，而不是設計協定，因此SDL的抽象資料型態（ADT）將變成設計者難以跨越的柵欄。此時，軟體設計者將陷入困境，他們不再擁有傳統程式語言的精確性，如C，並且市面上並沒有SDL編譯器或交叉編譯器，因此當在Target上實行SDL系統時，C或C++的程式碼產生都將會是必要的。然而，ADT的基礎概念無法直接轉譯成C或C++，某些特殊的運算子的產生，將會導致程式碼無法使用。

此外，整合前人所遺留的程式碼也是一大麻煩，因為SDL與C/C++的資料型態不同，需要重新建立兩者間的溝通橋樑，況且，當需要將SDL系統所產生的程式碼整合到RTOS上時，某些SDL的語意概念並無法直接支援。

我們可以看看下面兩個簡單例子：當將RTOS的優先權概念實施在任務（task）上時，SDL則將優先權交付給訊息（Message）。SDL語法認為無論任何時候，即使RTOS的執行被打斷，自動切換也不能被中止，尤其是在切換不同優先權的任務時發生系統呼叫。此時為了保證產生的程式碼能夠如同我們所預期地運作正常，系統會自動建立SDL虛擬機，但虛擬機使得程式碼與Target間的整合變得非常難以處理，並且根本無法存取使用某些RTOS服務。這一切災難都是由於SDL中並不存在信號（semaphore）的概念，但偏偏信號的使用在即時應用程式中是最典型的同步機制。由於SDL的起源是針對協定如何與遠端實體進行溝通，因此信號（semaphore）的概念在當時並不需要納入考量。

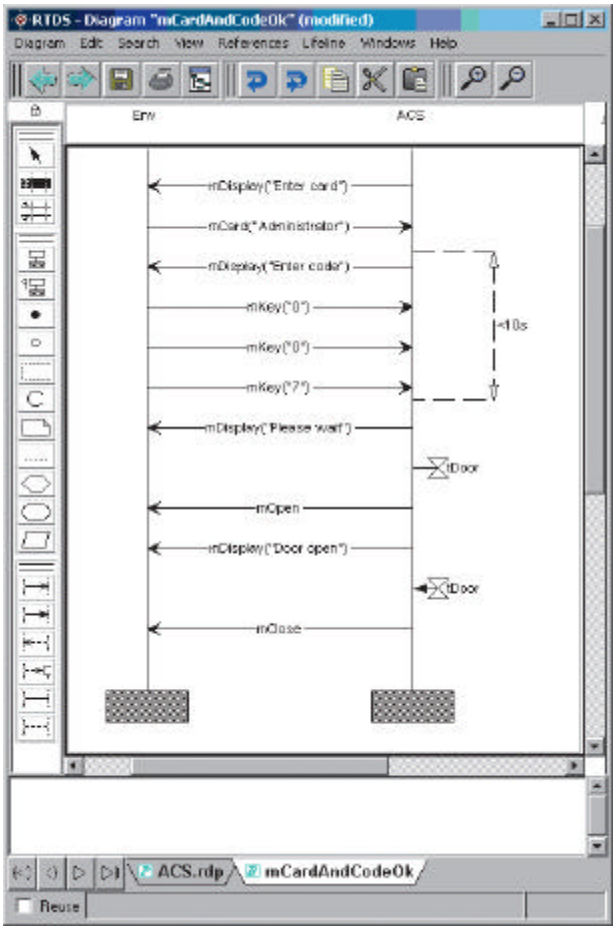

圖2. MSC規格範例

基於上述理由，為了要實行SDL中所描述的系統，工具將會變得非常難以製造、非常昂貴，並且需要很長的訓練時間，再加上超級棒的技術顧問，才有辦法使系統在Target上正常運行。最後終於，一切問題都解決了，我們才會發現：使用圖形化語言真是一點好處也沒有！

造成這樣情況的主因是，軟體開發者希望堅守SDL的標準，無論我們已經看到哪些必然會發生的技術問題。因此，SDL使用者開始在SDL圖表中寫入C，並打破SDL語法以符合未來技術。但既然商用工具無法支援C與SDL的交替使用（因為無法符合公定標準），這一批人何不發展他們自己的工具鏈呢？這也就是為什麼Alcatel、Nokia、Nortel、EADS和Sagem這些主要的電信製造商都開始發展他們自己的C code generator，以從SDL與C混雜的圖形化設計圖中獲得可用的程式碼。然而，第一版的SDL-RT就是看準了這樣的需求而發行的。為了擴展這種

語言普遍使用到所有即時應用程式中，SDL-RT也加入了圖形符號來處理信號。

UML（統一塑模語言，Unified Modelling Language）一正如其名－用以整合好幾種圖表。一開始的目的是為了將應用程式文件化，並且縮短軟體發展團隊中的人員溝通時間。UML第一版發行在1997年，得力於強大的銷售量，UML已經快速地普及到許多地區。該語言以其完整的物件導向方法為特色，基本上任何系統中的任何元素都可當作一個類別。這種通稱性的方法提供一個非常高階的抽象層，因此可以應用到任何概念上；但另一方面，UML並無法專用於任何應用領域，對於應用的細部描述也不夠精確。事實上，UML主要使用於設計初期的分析或規格定義階段。

UML圖表與設計之間薄弱的關聯性導致兩個不斷重演的問題：UML記號與最終程式碼之間的同步，以及缺乏組織架構以驅動發展程序。上述問題顯露了UML成功的事實，是由於軟體發展團隊依舊繼續做他們過去所做的，但使用這個語言來將程式碼文件化。這種做法的好處其實是非常有限的，這也就是為什麼某些大公司決定停止使用UML。在標準1.x版中所包含的九種圖表中，即時系統僅使用類別圖、序列圖，以及狀態圖。雖然UML基本是一個畫圖的工具，程式碼產生的能力很有限，但對業界來說遠比SDL工具負擔的起。

UML使得技術普及，但不可否認許多大型開發團隊密集使用UML的經驗，最後都是以失敗收尾。從上述情況看來，這些工具產生在Target上可運行的低效能、無臭蟲大型程式碼，幾乎指向同一個問題：在即時與嵌入領域中，使用一個完全物件導向的傳統方法真的是可行的嗎？這樣的問題是很難給一個直接的答案，但是顯然物件導向的發展能夠成功地針對系統模式的重用性（re-use），提供有效益的方式並納入資料庫的組織元件。再說，UML的市場力量太強大，因此SDL必須針對自己身為圖形化塑模語言的地位重新定義。最新的SDL版本為'2000，並且將UML的類別圖整合進來，為將來與UML2.0整合而鋪路。為了針對各種應用領域來定義專用的設定檔（profile），新版的UML主要的目標是支援MDA（模式驅動架構，model driven architecture）。

反觀SDL這邊，先不管軟體商們一開始的承諾，最新版的SDL語言依舊沒有實行到任何一個工具中。一般來說，因為SDL的複雜性，導致大部分的使用者傾向使用UML－即使精確性較低，也完全不為SDL所動。於是，SDL論壇開始成立一個特別小組來，將重新定義和簡化SDL的特性，並且在SDL中定義UML profile。此時，SDL使用者仍然使用'96版。

SDL-RT 2.0版整合了UML的類別圖與部署圖，不但可補足SDL的圖表顯示，亦可保持其在不同表示法中原有的一致性。UML 2.0也整合了SDL中現有的特點，像從區塊圖到結構圖，以及MSC到序列圖，也可以定義專門領域的設定檔，但不幸地，並沒有針對標準的即時設定檔。因為那些在OMG擁有投表權的人員職務調度，主要業界公司都質疑這樣的設定檔到底有沒有標準化？

然而，第一版的UML2.0目前已經上市，新標準的最初目的是為了簡化程式的移植性與可讀性，但每個制定者都只專注於自己專用的設定檔，而使得目標並未達成。這些設定檔不適合共用，工具本身也很難提供使用者自行修改設定檔，因此最後市面上充斥一堆各種版本的UML2.0工具，對使用者來說，程式碼的閱讀與移植比以前更加困難。

無論工具本身是以SDL或UML為基礎，規格制定者們內心都有一個預言：透過研討會和廣告，並用他們的語言與所花費的精力來證明他們是對的。這樣的行動引起的許多技術狂熱者，並且他們再也無法聽進其他人的技術建議，直到技術實務將他們又帶回了最初的思考，但代價是什麼？多想想，我們所需要的就是像SDL-RT這樣的工具，能夠透過工業用途與理想的啟示，提供實務的解決方案，而不須忽略目前的趨勢。這就是為什麼使用者需要把某些UML圖表、大部分SDL圖表，以及標準程式語言C/C++放到同一個一致性的架構中：目的就是為了針對即時與嵌入領域，取出各種不同語言的所有優點，缺點都丟

掉。為了能夠重複使用這些系統圖表，而不須使用專門的工具，所有的資料都會儲存在XML檔案中。拋開語言本身不談，軟體工具的品質必須好到足以說服使用者，雖然開發團隊的目標無法與現有標準一致，但SDL-RT可以，並且帶給系統設計最佳的品質與效能。

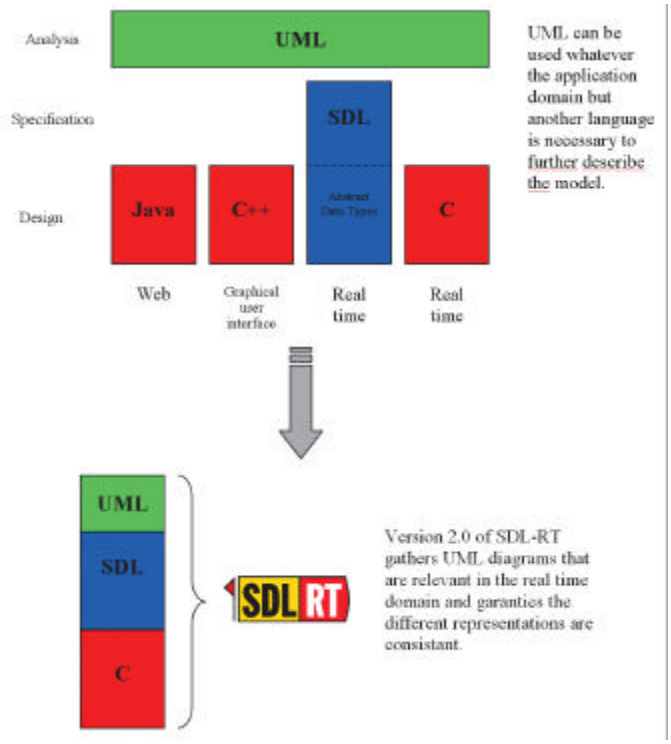

圖3. SDL-RT2.0版包含了UML圖表，並且確保原有的一致性。