ATK (HTK 應用軟體套件)

譯者：翁大程

簡介

ATK 是以 HTK 爲基礎，可用於開發實驗性應用程式所設計的一組 API。它是 HTK 標準函式庫上層的一組 C++中介層所組成。這個中介層可以讓目前的語音辨識開發者選擇不同版本的 HTK 並與 ATK 一起編譯後來開發應用程式。如同 HTK 一般，ATK 也是具有跨兩種作業系統平台(Unixs 與 Windows)的特性。

ATK 是一種多執行續程序(Multi-Threaded)。她允許多位使用者(User)的輸入，如語音命令、滑鼠按鍵、手寫輸入等，並將這些輸入合併爲單一資料串流，另外，也允許多個資料串流與辨識器。透過調整執行緒優先順序以及效能可減少延遲(Latency)並且提高系統的反應能力。

1.1 ATK 核心(Core)

ATK 基於三種基礎 C++物件類別，分別是封包(Packet)、緩衝區(Buffer)與元件(Component)。以下說明。

1. 封包: 封包是指一串包含資料的記憶區塊。當執行元件(如 Component 說明)時，許多封包會以分同步的方式在元件中傳送不同的資料。在某些特定的封包中，會包含像使用者輸入或是輸出資料(如語音或滑鼠事件等)。在上述情形中，每一個封包會包含一個時間標籤(Time Stamp)，用來定義相關的暫時跨距資料來源(Temporary span)。每一個封包的資料型態可能會是文字片段、語音片段、編碼過的特徵向量、文字標籤、或是語意標籤等。

2. 緩衝區 ：ATK 的緩衝區是一種 FIFO(先進先出)的封包佇列(Queue)。緩衝區提供了一個通道，用以在元件間傳遞封包。緩衝區可以限制其大小或不限制其大小。在存取緩衝區時，ATK 會先利用一種機制先測試該緩衝區是否正在運作中，若是、則會將該緩衝區先 Block，等運作完畢之後再解除 Block，屆時就可以執行存取動作。

3. 元件：真正處理的元素或單元。每一個元件在執行時，擁有自己的執行緒。三種 ATK 最主要的元件分別爲音訊來源元件、編碼元件以及辨識元件。元件間的溝通借由 Buffer 通道傳遞封包。每一元件有一個命令介面在執行階段時用來更新控制參數，藉此來修正該元件的行爲。舉例來說，音訊來源元件會

因爲不同的需求而停止或開始取樣。

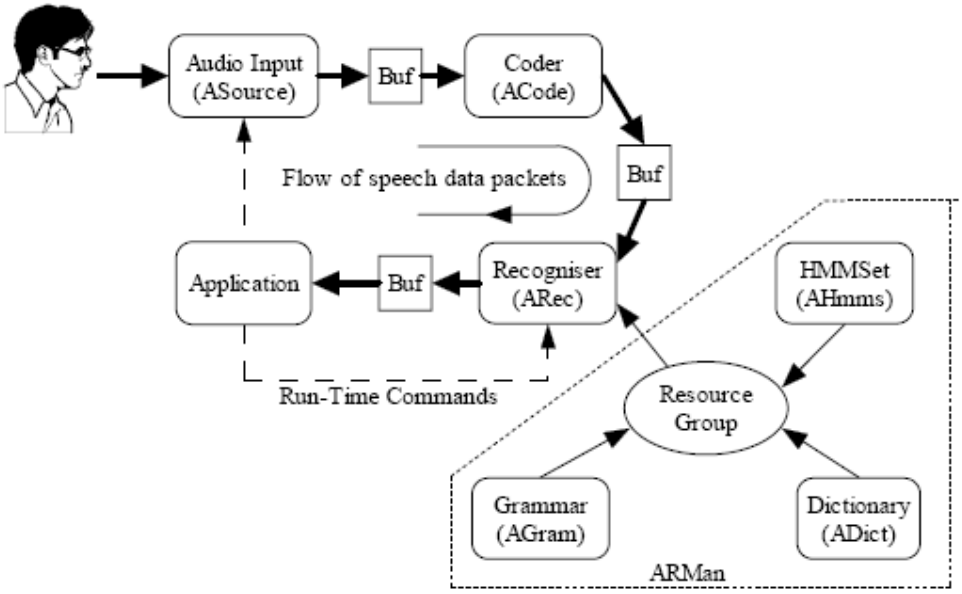

Figure 1 - Basic recognition system

圖一表示一種利用 ATK 的基礎系統。應用程式與音訊元件溝通來開始與停止聆聽且和辨識器連結來控制辨識過程。另外一種典型的情形是應用程式會預先在每一次使用者語料前載入一個新的辨識文法。

辨識器元件本身會依附在不同的辨識來源物件。在ATK中有三種主要來源物件：字典(ADict)、文法(AGram)以及隱藏馬可夫模型組(AHmms)。字典物件定義每一種字彙項目如何發音的方式，這個物件通常會以讀入外部的檔案的方式來初始化。文法物件定義一組可允許字序列的有限狀態網路。每一組有限狀態網路可被定義爲所謂的較低階層之子網路，任何字或子網路均可被賦予一種語意標籤。字與字之間的轉換亦可賦予機率值，除外、N-Gram的語言模型可以加在第四種額外的資源物件型態(此資源稱爲ANGram)。隱藏馬可夫模型組包含用在語音模型的隱藏馬可夫模型，而且該隱藏馬可夫模型組也是由從檔案來初始化。

以上所提隱藏馬可夫模型組、字典物件、文法物件和N-Gram語言模型都是儲存在資源資料庫中，並且由辨識資源物件(ARMan)來管理。管理的行爲分別是儲存、編輯與刪除。當辨識器被執行時，其運作是依賴儲存在特定資源群組的資訊編譯成的網路而定。應用程式可以依照需要來改變資源群組。

註一、HTK 標準函式庫已經被擴充用來支援 ATK 需要的額外的功能，這項額外的功能就是可以讓 atk 編譯後使用，此外，有一項新的 HTK 模組稱作 HThreads，

它提供支援執行緒的基本平台。
註二、支援 MFC 的 Framework 並不表示支援跨平台的 GUI，也就 HGraf 的函式庫僅僅被擴充爲支援多視窗而非支援跨平台的 GUI。
註三、ATK 亦支援 Bigram 以及 Trigram 語言模型。

在執行階段時，辨識器可以在不同的組態模式中運行。這些模式可以在辨識器啓動或停止時控制狀態，即便是在自動或手動重新啓動的情況下、或是控制回傳結果的方法。在理想假設的狀況下，辨識器會及時將辨識結果回傳至 Shell 程式輸出畫面，以及在語料節被偵測爲結束下以批次方式呈現。辨識器亦可回傳互動式圖表。

元件的初始組態由一個全域的組態檔案所決定並在程式開始時被讀入。這個檔案亦可被用來組態 HTK 函式庫。一旦元件物件被執行後，所有組態必須透過執行階段的命令介面來改變。

舉一個較複雜的組態爲例，該系統有兩個辨識元件以及一個分離語音與靜音的偵測器(用來更明確的過濾資源)。使用兩個辨識元件可以驅動其中一個辨識器用來當辨識一句關鍵片語或者是其中一個辨識器可以用來處理少量的字。當系統是設計爲在本機中以語音瀏覽網站，此時必須由外部來源載入文法時，這樣的作法是相當有效。但是，必須要考量的是如何利用有效可靠的方法回復到本機命令模式。在這樣的範例中，也就是有額外的輸入來源情況下，封包必須錄製滑鼠動作。最後一個封包必須完整被系統傳送到應用程式中。這種形式的架構必須能支援多模式語音輔助封包(multimodal voice-assisted CAD package)，在這種封包中也許是以“Draw a line from here <click mouse> to here <click mouse>”的形式呈現。

最後必須要注意的是，雖然在此會專注在單一 SSDS 介面系統的範例。然而 ATK 的目標是提供一組使用 HTK 衍生出來的辨識資源的即時應用程式的建置介面。

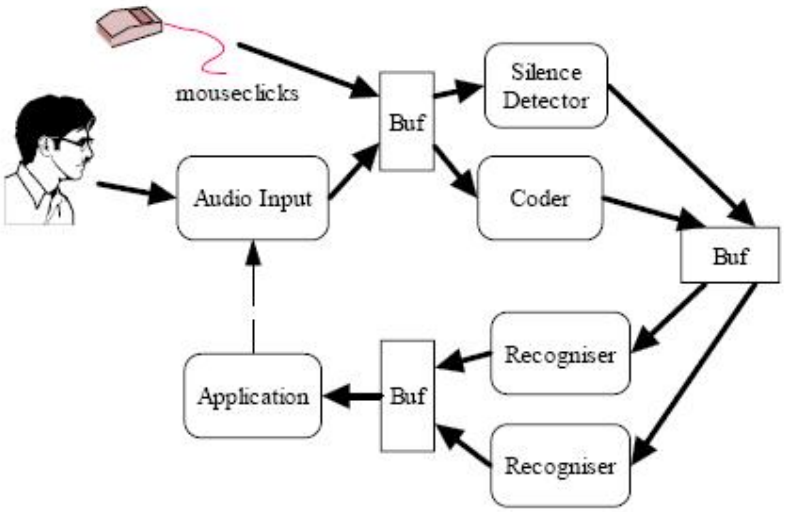

Figure 2 - A more advanced recogniser configuration

圖二、一種更進階的辨識器組態

1.2 在應用程式中使用 ATK

對許多簡易的應用程式而言，針對 ATK 來撰寫介面時，必須考量的是恐怕不只是元件物件的初始話或者是處理辨識器中迴船的結果。簡言之，以一個非常簡單的數字辨識系統來說，它也許是使用不必動手播的撥號系統。在這種情況下，需要的字典與文法可以預先被離線準備。字典檔中應該要具備每一個數字的拼音。

```
ONE         w ah n
TWO         t uw
etc
```

文法檔中應該要定義簡單的數字迴路如同圖3所示。依照 HTK SLF的格式，文法檔應該包含文法節點(也就是字)的列表，以及用一迴路將列表連結起來。以下片段描述基本的觀念。

```
VERSION=1.0
N=16  L=25
I=0    W=!NULL
I=1    W=SIL
I=2    W=!NULL
I=3    W=!NULL
I=4    W=ONE
I=5    W=TWO
...
I=15   W=!NULL
I=7    W=OH
J=0    S=0   E=1
J=1    S=1   E=2
J=2    S=2   E=4
...
J=24   S=14   E=15
```

第三種需要的資源是一組隱藏式馬可夫模型。這些資源是需要使用 HTK 離線先準備好。在這種狀況下，通常是 HMM.list 需要記錄模型列表，HMM.mmf 需要包含語音聲學模型。

註五、HTK SLF 的格式在 HTK Book 中有詳細說明。

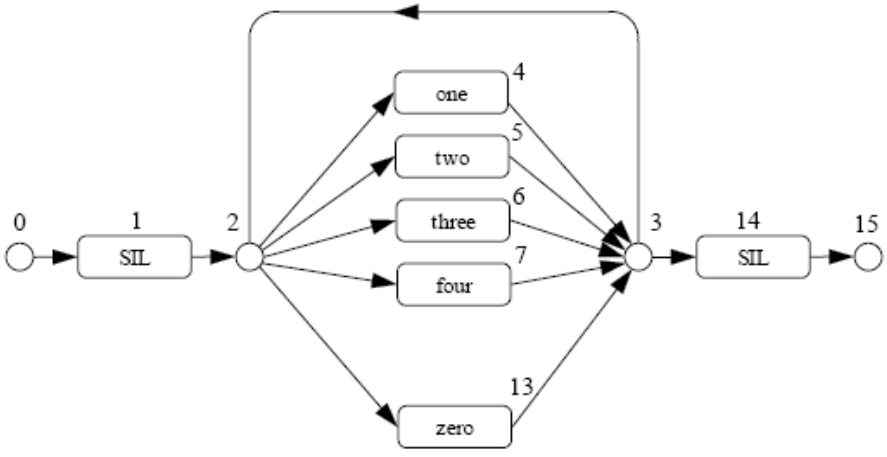

Figure 3 - Simple digit recognition grammar

以上三種資源可以在程式啓動時，置放在組態檔內的項目裡定義。在組態檔內通常也會包含編碼參數的規格，以下爲組態檔的範例。

```
# define coding/feature parameters
    SOURCEFORMAT        = HAUDIO
    SOURCERATE          = 625
    TARGETKIND          = MFCC_E_D_N_A
    TARGETRATE          = 100000.0
    USEPOWER            = F
    NUMCHANS            = 24
    CEPLIFTER           = 22
    SILFLOOR            = 50.0
# define needed resources
HMMSET:   HMMLIST       = "HMMS.list"
HMMSET:   MMF0          = "HMMS.mmf"
ADICT:    DICTFILE      = "dialer.dct"
AGRAM:    GRAMFILE      = "dialer.net"
```

請注意在 HTK 中模組的名稱是以物件的名稱而非類別名稱。將所需不同的資源組合起來時，所要撰寫的程式以直列式即可。主程式中必須要先將 ATK 初始化，呼叫在 HTK 函式庫中的 InitHTK 並指定版本。以下是初始化範例。

```
char *version = "!HVER!MyApp: 1.1.39 [SJY 31/02/05";   //先指定版本
if (InitHTK(argc,argv,version)<SUCCESS)
{
    printf("Error: cannot initialise HTK\n"); exit(-1);          //初始化失敗
}
```

不同的物件也是以這樣的方式初始化。整個架構如同圖 1 所描述。第一步先定義不同的緩衝區。

```
ABuffer auChan("auChan");          // 波型資料緩衝區
ABuffer feChan("feChan");          // 特徵緩衝區
ABuffer ansChan("ansChan");        // 辨識結果緩衝區
```

然後將初始化來源與編碼器元件，並連接正確的緩衝區。

```
ASource ain("AIn",&auChan);                    // 將auChan 由來源連結到編碼器
ACode acode("ACode",&auChan,&feChan); // 將feChan由編碼器連結到辨識器
```

在辨識器初始化之前，資源管理器必須先被建立且將不同的資源儲存進去。

```
AHmms hset("HmmSet");     // 建立一組HMM並命名爲HmmSet
ADict dict("ADict");          // 建立一個字典資源並命名爲ADict
AGram gram("AGram");      // 建立一個文法資源並命名爲AGram
ARMan rman;                    // 建立一個資源管理器
rman.StoreHMMs(&hset);    // 將資源存入rman資源管理器中
rman.StoreDict(&dict);
rman.StoreGram(&gram);
```

請注意在組態檔中必須先定義這些資源的真實檔案名稱。接續可將所有資源群組化然後再出始化辨識器。請參閱以下範例。

```
ResourceGroup *group = rman.NewGroup("dialer"); //新增群組並命名爲dialer
group->AddHMMs(&hset);                          //將資源加入群組
group->AddDict(&dict);
group->AddGram(&gram);
ARec arec("ARec",&feChan,&ansChan,&rman);     // 建立辨識器
```

請注意建立建立辨識器時必須將正確的緩衝區由編碼器連結到辨識器，最後應用程式方能讀取辨識器輸出結果後的緩衝區。

最後的步驟是啓動元件。

```
ain.Start(); acode.Start(); arec.Start();
```

在預設情況時，辨識器會處在閒置狀態當初始化後，除非以送出命令訊息的方式將閒置狀態改變爲開始狀態。

```
arec.SendMessage("start()");
```

請注意這裡的"start()"爲一個 ATK 執行階段的命令，表示該命令可在執行階段中，被目前正在接收中 ARec 元件直譯出來，這些跟 C++的程式碼沒有關係。一旦辨識器開始從編碼器中接收封包，它會不斷地將辨識結果送回應用程式。而這些辨識結果是由 ansChan 緩衝區中抽取出來的。以以下的範例程式來說，以一個無窮迴圈可以將封包由終端機裡(A Shell Program such as DOS in Windows O.S) 列印出來。在實際的應用程式中應該會做更複雜的辨識處理。

```
while(running)
{
    APacket p = ansChan.GetPacket();
    p.Show();
}
```

最後，ATK提供一組監控元件用來監控與控制其他的元件行爲。若要是使用此一監控元件必須在其他元件被啓動前先出始化監控元件。以下爲程式範例。

```
AMonitor amon;                  // 建立amon監控器
amon.AddComponent(&ain);        // 將要監控元件加入
amon.AddComponent(&acode);
amon.AddComponent(&arec);
amon.Start();                   // 啓動amon監控器
```

如同以上所描述，使用 ATK 來製作一個簡易的應用程式僅僅需要少量的程式碼來實現。在實際的工作上，以上面所示範的程式碼來延伸的主要部份將會是改變文法來辨識每一句語料。簡單的方法就是預先載入多個文法檔，然後變換它。舉例而言，若一個應用程式需要三種文法檔，我們可將他們指定到不同的資源群組。

```
ResourceGroup *group1 = rman.NewGroup("group1");
group->AddHMMs(&hset); group->AddDict(&dict); group->AddGram(&gram1);
ResourceGroup *group2 = rman.NewGroup("group2");
group->AddHMMs(&hset); group->AddDict(&dict); group->AddGram(&gram2);
```

```
ResourceGroup *group1 = rman.NewGroup("group3");
group->AddHMMs(&hset); group->AddDict(&dict); group->AddGram(&gram3);
```

請注意每一個資源群組共用相同的HMM組以及自點資源，但是個別擁有自己的文法資源。以上面的狀況來說，辨識器的文法可以藉由命令介面送出命令來置換文法。

```
arec.SendMessage("usegrp(group2)");
```

當辨識器接收到這樣的命令時，並不會即刻反應，而是在下一語句辨識時才會置換文法。以上簡述文法的用法。ATK允許多個文法物件以並列方式存在在同一個資源群組中，亦可允許文法物件被複製、儲存、回存、編輯與建立。本章節是針對什麼是ATK與如何使用ATK兩方面的議題。在手冊的其餘章節中會對ATK的核心元件類別、封包、緩衝區等有詳細說明。亦會針對系統元件如音訊來源器、編碼器、辨識器、資源、資源管理器做說明。其中也會有一章節提及ATK與HTK相關議題如編碼的差異、使用慣例、支援資源等。最後會以一些範例應用程式來做說明，其原始碼也會附在ATK轉發套件中。