國立交通大學

資訊科學與工程研究所

碩 士 論 文

針對數位電視機上盒設計有效率之 DSM-CC 解碼器

Design of an effiecient DSM-CC decoder for DVB-MHP Set Top Box

研 究 生：丁健文

指導教授：蔡淳仁　教授

中 華 民 國　九 十 五　年 十 月

針對數位電視機上盒設計有效率之 DSM-CC 解碼器
Design of an efficient DSM-CC decoder for DVB-MHP Set Top Box

研 究 生：丁健文　　Student：Chien-Wen Ting

指導教授：蔡淳仁　　Advisor：Chun-Jen Tsai

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of Master

in

Computer Science

October 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年十月

摘要

本論文的研究重點，是在嵌入式平台上發展 DSM-CC decoder，以提供數位電視機上盒的模擬系統使用。目前我們能夠找到的分享軟體都是只有 middleware (MHP) 的實作，但是要加上 DSM-CC decoder 的部份才能夠成其爲一個完整的數位電視機上盒的模擬器。在本論文之後的章節中，會介紹整個系統的設計概念及背景知識，以期釐清在規格書當中，大量且複雜的觀念。之後再介紹整個系統在設計與實作上的特色，並且驗證能夠正確的在嵌入式發展平台上執行。要如何快速的分辨我們收到的每一筆資料，送給相對應的解碼器處理，並產生正確的資料格式再加以儲存是我們討論的重點。

Abstract

Most of the open source multimedia home platform (MHP) simulators just design the architecture of DVB-MHP and implement a little of the core code. But what we need is a complete Set Top Box simulator, which is needed to add at least a DSM-CC decoder and some other components. In our thesis, we will discuss with the detail knowledges in the DVB-MHP specification first to make readers to have a clear concept of the DSM-CC decoder. And then, we will describe the detail implementation steps and the ideas why we have to do this. At last, we will show the correctness of our implementation by some interesting experimentations. How to recognize the data type we received, parse them, and send them to the next decoder efficiently is the key point in our implementation.

章節目錄

List of Figures

List of Tables

1. 簡介

數位電視廣播系統（Digital Video Broadcast System）具有高影像解析度，高傳輸品質，以及提供加值服務的特性。爲了能讓數位內容產業界能夠在一個相容可互通的平台上提供加值服務， DVB 組織從 1997 年開始發展 DVB-MHP (DVB Multimedia Home Platform)的工作項目，來訂定一組標準中介軟體（middleware）以提供加值服務應用程式所需的繪圖、影音控制、服務資訊等等的界面函式。數位電視已逐漸成爲目前世界上廣播電視媒體的潮流。我國對於數位電視的發展更是不遺餘力，並預定 2007 年將達成全面廣播電視數位化。

因爲以 MHP 作爲 middleware 具有讓一般的開發者可以在開發程式時忽略底層硬體的差異、提供給一般應用程式的開發者一個標準的平台，並且處理了大部份較底層跟 broadcast stream 的讀取及繪圖相關的部份，讓使用者更容易在 middleware 之上開發軟體，這是我們採用 MHP 的理由之一。

此外，因爲 MHP 的規格書是完全免費且開放的，只要是有興趣的人，都可以自由的下載並且獨立開發自己的 MHP。又因爲數位電視的應用程式都是用 Java 或是 html 撰寫的，所以不會因爲底層的硬體的不同，而要針對硬體開發不同的應用程式，因此 MHP 在規格書上對於硬體並沒有十分嚴格的要求，在這部份更是給使用者相當高的自由度，這也是我們採用 MHP 的理由。

藉由數位化的傳播方式以及使用者端的數位電視機上盒之接收、解碼、與播放，使得數位電視具有提供、並在使用者端執行應用程式的能力。數位廣播與接收系統的發展已臻成熟，因此數位電視機上盒 STB(set top box)的解碼與執行能力將直接影響使用者端應用程式服務之效率。在 MHP 標準中，是採用由 MPEG 所制訂的 Digital Storage Media Command and Control(DSM-CC)通訊協定來負責應用程式和資料的傳輸，DSM-CC 相關功能之設計與製作是本論文重要的目標。爲了將應用程式由提供者端（電視台）發送出來，應用程式被切割並包裝成可經

由廣播傳送的 MPEG-2 Transport Stream，如何從 MPEG-2 Transport Stream 之中有效率地抽取出其中的 DSM-CC stream，來下載 MHP 應用程式至 STB 中，對 STB 的效能有很大的影響。本論文將簡介 TS 的產生方式，並且將 Transport Stream Parser 列入本論文實作討論的部份。由於應用程式相關的資訊皆經由 Application Information Table(AIT)傳遞給接收端，快速而有效率解出 AIT 的內容才能保證 STB 的效能，再配合快速的 DSM-CC 解碼，可以更快速的讓使用者開始選擇並播放加值服務的應用程式。

本論文由 transport stream 的接收階段開始，以實作的方式探討如何有效率並且正確的從其中抽取出我們所需要的一切資料，並且把它一層層的傳給每一個階段的處理器，直到解出檔案的原始狀況並且儲存到記憶體或是硬碟當中。

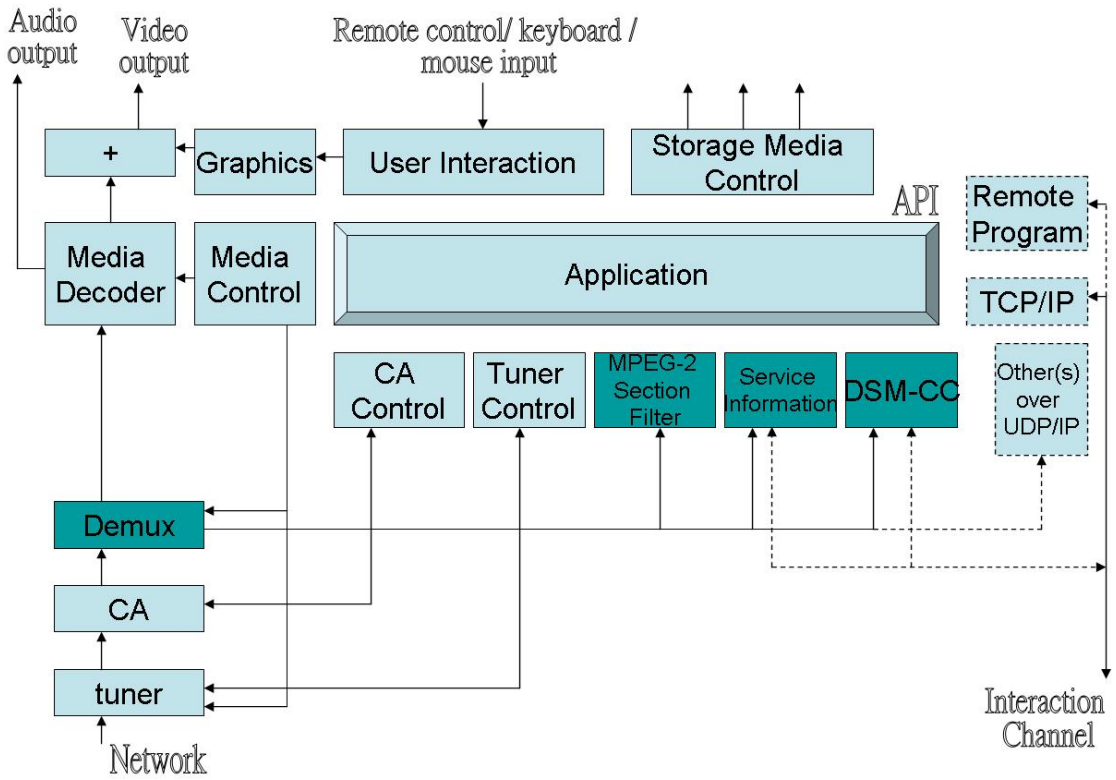

Figure 1. Interface between mhp application and mhp system

爲了確實達到MHP規格所訂定的要求，我們藉由參考DVB-MHP的規格[1]書、13818-1 的規格書[2]、13818-6[3]的規格書，作爲本論文設計實作之DSM-CC及數位電視管理系統其他部份的正確性、可行性、以及執行效能的根據，並藉由

處理目前在台灣實際播放的數位電視串流，來驗證本論文之實作的正確性。

根據數位電視管理系統各部份的功能與特性，本論文將分別探討下列 MHP 相關的功能：

1) MPEG-2 傳輸層：transport stream(TS)的產生、傳送、與接收之後的處理。

2) 應用程式及資料的接收與解碼：digital storage media command and control (DSM-CC)的子系統以及各部份的處理。

3) 應用程式管理器（Application Manager）相關模組：

   A. Program Association Table(PAT)及 Program Map Table(PMT)的處理器。

   B. Application Information Table(AIT)的處理器。

根據我們所提出的架構所實作的系統，可確實達到高效能的數位電視應用程式之接收、解碼、以及播放。

本論文下面幾個章節的組織如下。首先，在第二章，我們會先對相關的研究進行討論。接下來，在第三章，我們會對系統的設計概念和背景做一個介紹。第四章則是描述實作的細節，而第五章則是討論實驗的結果。最後，在第六章會對一些未來可能進行的系統改進方式進行討論。

2. 相關研究

隨著數位家庭時代的到來，越來越多的人力及資源投入這個市場中。而因爲數位多媒體壓縮的技術成熟以及傳播網路的發達，更使得數位電視機上盒(STB, Set Top Box)的開發，成爲其中相當重要的一部份。根據 MHP 的規範，加值應用程式的開發語言有兩種，分別是 Java 和 HTML，而 Java 應用程式則是所有符合 MHP 標準的機上盒都要能執行的。目前業界實作 MHP 機上盒的作法多數爲在底層的硬體及作業系統之上，建構 Java Virtual Machine，並且在 JVM 上提供跨平台的 MHP 界面程式庫，所有的應用程式都在 MHP 之上執行。由於現有系統的基本架構相似性高，所以影響整體效能的主要原因除了不同 VM 的選擇之外，就是在於 MHP 的開發實作上的不同產生的差異。本章將會就一些不同架構的實作或是開放原始碼的 MHP 加以討論。

2.1. OpenMHP

目前開放程式碼的MHP相關實作以OpenMHP[4]最爲完整，也被廣泛應用於開發MHP相關的系統與應用程式。

OpenMHP是由Tejopa的研發團隊，針對MHP的specification所提出的一套開放原始碼的MHP模擬執行環境，目的是爲應用程式開發者提供一個測試的環境，OpenMHP的架構如Figure 2所示，主要可以分成三大部份。

1) Application Manager，用來控制管理所有目前 OpenMHP 可以執行的應用程式以及接收使用者的輸入。

2) OpenMHP的主架構，也就是middleware 的部份，包含了DAVIC、DVB、HAVI這三個部份，不過其中除了HAVI的user interface大概有一半已經實作完成，DVB跟DAVIC除了跟一些基本功能相關的部份之外，剩下的都還沒有實作。至於MHP規範中的Sun JavaTV API，在Figure 3、Figure

4、Figure 5當中，我們分別列出了OpenMHP中DAVIC、DVB、HAVI的架構，跟這些主要的classes的功能，由於這不是本論文主要探討的方向，所以細部的實作情形，就不加以列出了。

3) 和硬體相關的adaptation layer的部份，如Figure 6。這部份的功能多數都要使用到硬體環境，不過OpenMHP在這部份是完全沒有實作。

OpenMHP原本設計的執行環境是Microsoft Windows，所使用的Java Virtual Machine是Java™ 2 Platform, Standard Edition。但是Windows並不適合在嵌入式的環境上執行，所以我們必須將OpenMHP移植到linux base的環境上；J2SE更是不符合DVB-MHP的規範，像J2SE中繪圖用的class library：javax.swing在OpenMHP的GUI中大量被使用，但是在DVB-MHP所規範的虛擬機器Java™ 2 Platform, Micro Edition當中，完全沒有這部份的實作，所以我們將來要把OpenMHP移植到嵌入式環境時，也必須把這部份重新實作。

由於 OpenMHP 對於較底層的 transport stream 的接收及解碼、Section Filter 及 DSM-CC 的處理等和硬體相關的部份並沒有實作。所以我們爲了讓使用者可以測試整個 transport stream 的包裝是否正確，以及整體執行的流程是否無誤，特別針對這部份開發了一個可以整合到不同系統的 MHP simulator、可以獨立執行、甚至可以配合 STB 使用的軟體環境。

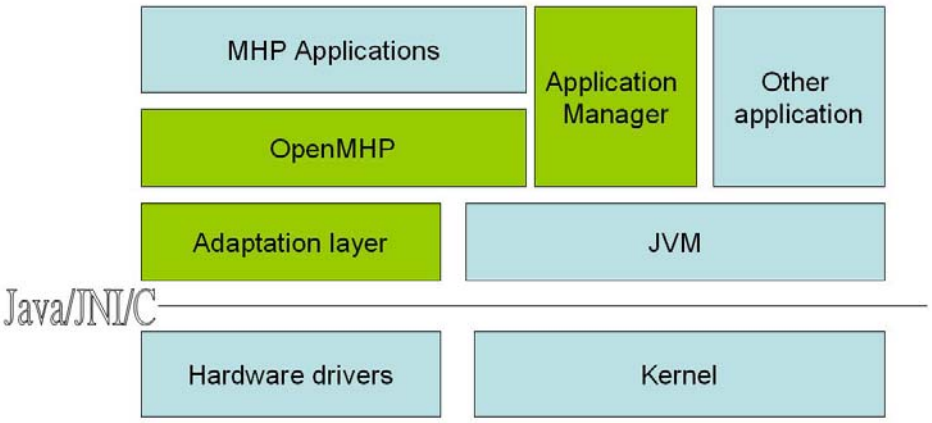

Figure 2. Base Architecture of OpenMHP

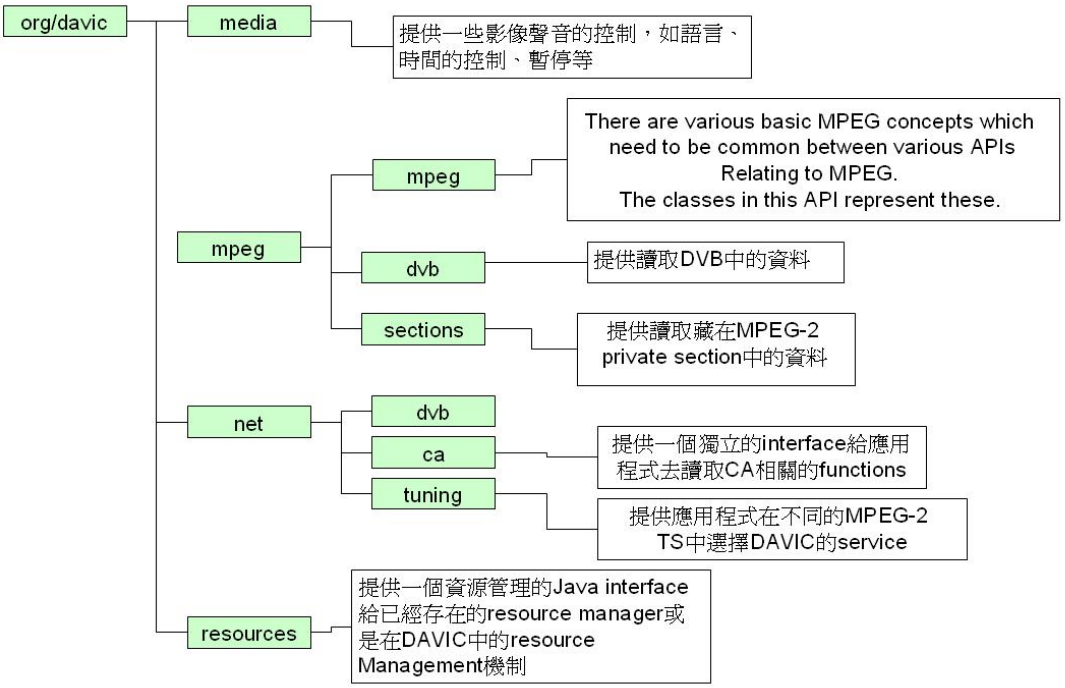

Figure 3. Architecture and Functions of DAVIC

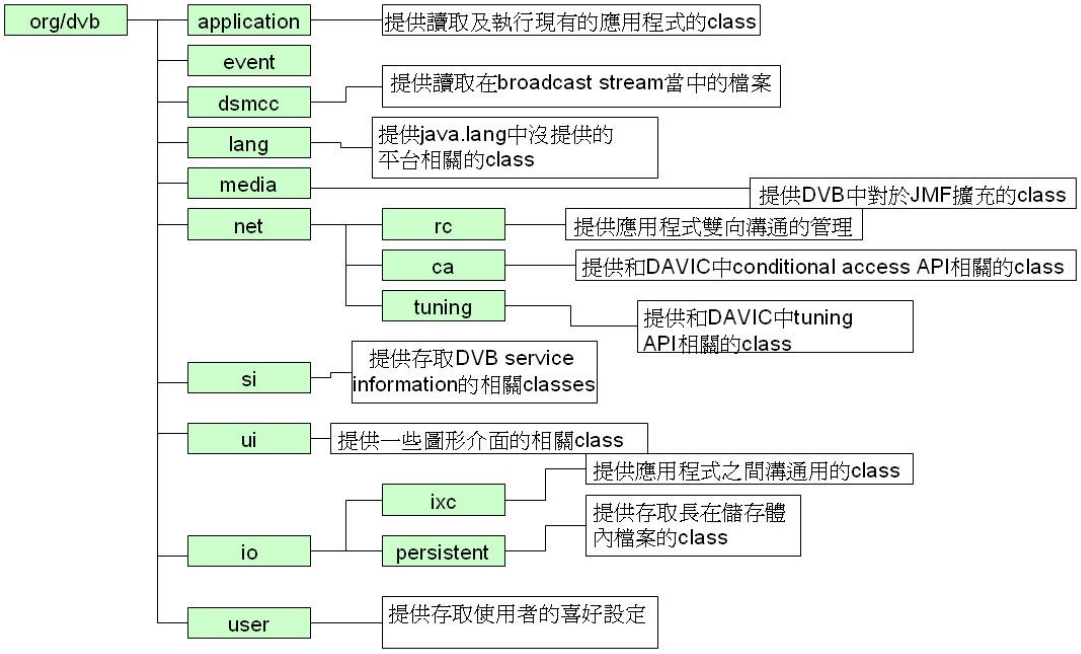

Figure 4. Architecture and Functions of DVB

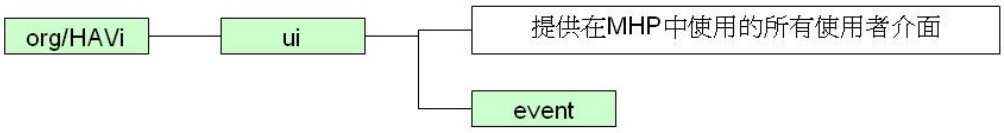

Figure 5. Architecture and Functions of HAVI

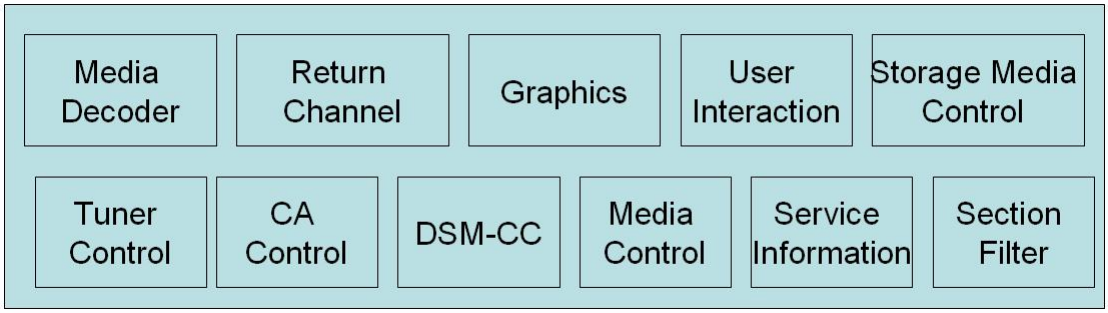

Figure 6. Adaptation Layer of OpenMHP

2.2. Section Filter - Two steps of filter and cache usage

當 STB 接收到廣播傳送的訊號時，影音資料經過 DEMUX 之後直接呈現在螢幕上，而 Applications 以及相關的資訊則必須另外處理。負責篩選出 applications 以及相關資訊的功能模組即爲 Section Filter。Section Filter 的設計會直接影響 Applications 與相關資訊接收的完整性以及後續處理的效能。

Section Filter接收到資料之後就依據資料特性將其送往適當的模組進行下一步處理，因此接收與送出資料的速率是主要考量。Hongguang Zhang等人在[5]提出一個Section Filter的架構，當Section Filter Task收到資料之後，並不馬上處理而是先丟到Section Buffer去，然後就接著接收下一筆資料。當Section Buffer空間將要不足時，就發出訊息給Protocol Parse Task，由Protocol Parse Task開始處理在buffer當中的資料。此外，Monitor Task負責監控整個處理的流程跟資料的Version是否有更新。所以Section Filter可以採用多工執行緒的作法，一邊接收新的資料往Section Buffer存放，一邊將Section Buffer當中的資料取出來處理並存放到cache當中。主要架構圖如Figure 7所示。

因爲我們整個處理的效能，會受到 Section Filter 接收、整理資料以及往適當模組傳送的速率影響。在 Hongguang Zhang 的實作中，雖然他避免了讓 Protocol Parse Task 一直作忙碌等待(busy waiting)的情況產生，只是卻也沒有提到當一份資料被接收時，該如何快速的分辨是不是已經重覆接收了，進而判斷是不是該作進一步的處理，所以這部份是我們該思考改善的。

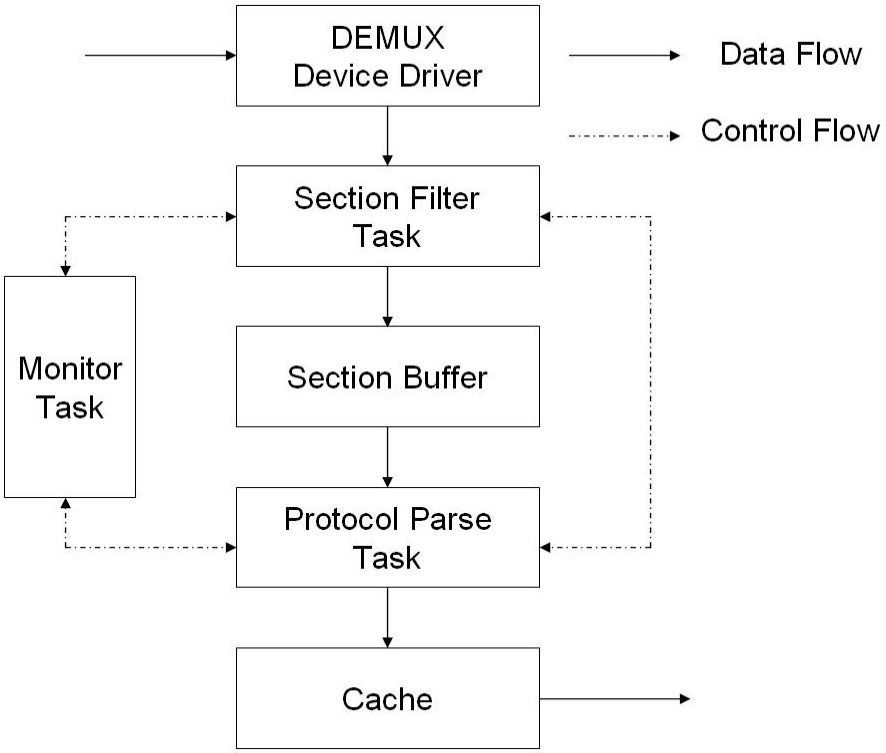

Figure 7. Running flowchart of the Filter module

另外，Hongguang Zhang等人還提出了一種儲存資料的模式。每一次broadcaster端送出來的資料，我們可以視爲一個carousel，在carousel node當中記錄了這個carousel的相關訊息，包括transactionId，version number之類的。一個carousel當中會有一個或多個的應用程式，分別將它們的訊息記錄在GourpNode，如number of modules、block sizes。每一個應用程式由多個moduels組成，在moduleNode當中記錄了moduleId、number of blocks等訊息。modules由blocks組成，blockNode記錄了基本的資料內容。綜合以上所述，Zhang的儲存格式就如Figure 8所示。

他們這麼作的目的是當使用者選擇不同的應用程式時，application selector 可以很快的透過 carousel node，groupNode 找到使用者所選擇的應用程式並且將它讀取執行。

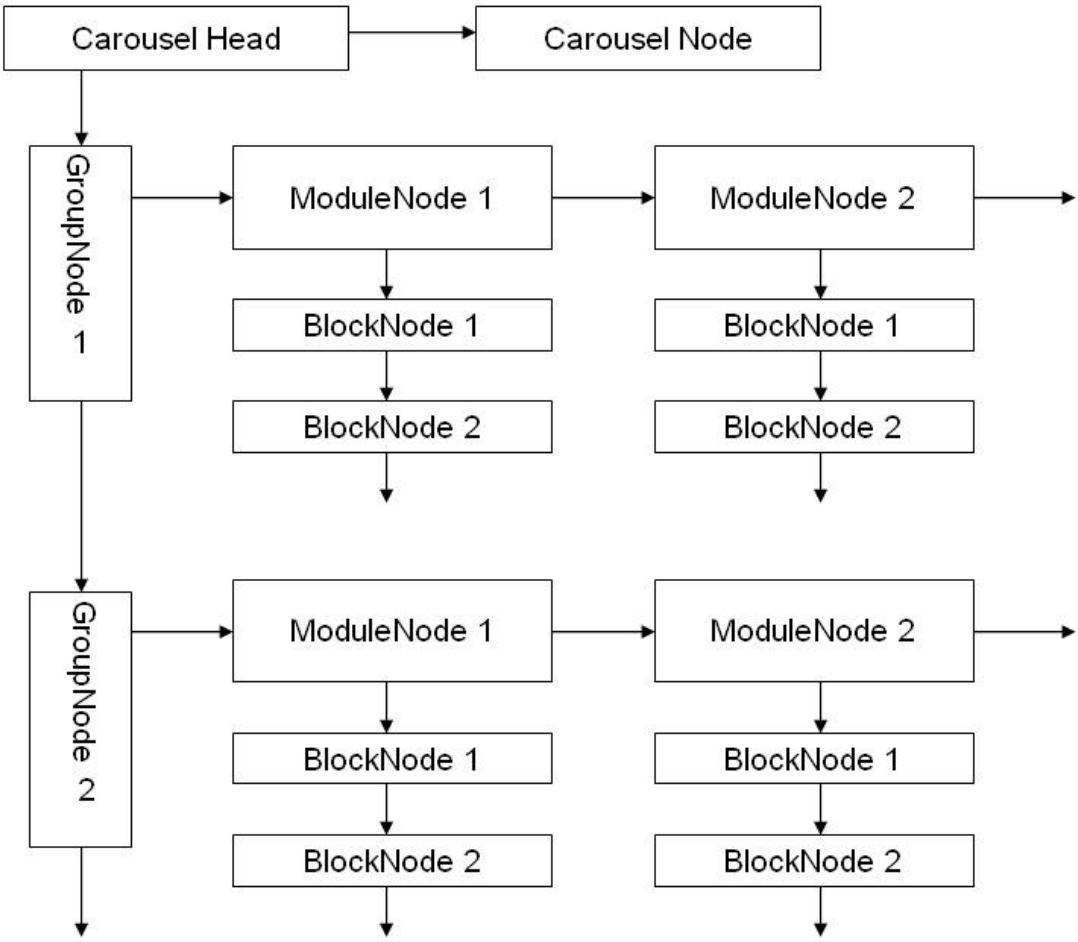

Figure 8. Storage architecture of caching data

2.3. Object Acquiring Time Saving Scheme

因爲 Object Carousel 當中包含的是 STB 可執行的影音以及應用程式，其中的資料是否能夠正確快速的接收並處理，會直接影響 STB 將訊息呈現給使用者的速率。爲了達到近似於即時影音及應用程式的執行，並且維持其資料的正確性，我們必須要能夠隨時掌握 Object Carousel 的最新狀況。

Eun-Jung Kwon等人提出了一個新的DSM-CC object carousel的產生方法[6]，可以節省接收資料的所需時間。在file system中取出需要的object之後，首先

先由object change inspector檢查這個object跟之前送出去的內容是不是相同，以及有沒有必要重新產生object carousel，如果有必要的話，就通知object classifier將這些objects分類成有更新過及沒有更動過兩種，再由module builder分別產生modules，之後的過程就和一般標準的DSM-CC object carousel產生相同。如Figure 9所示。

而接收者端的接收並解碼的過程則加入檢查modules是不是已經被處理過，檢查module version是不是有所更新，以及針對特定的modules作decode的動作，詳細的流程圖如Figure 10所示。

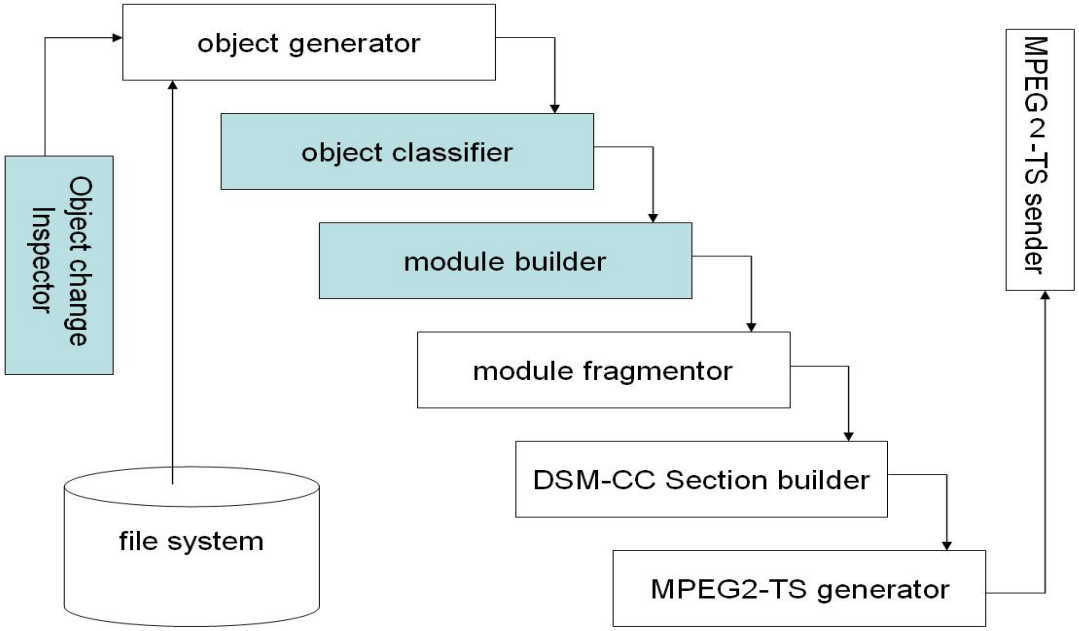

Figure 9. The Process of Kwon's DSM-CC data encoder

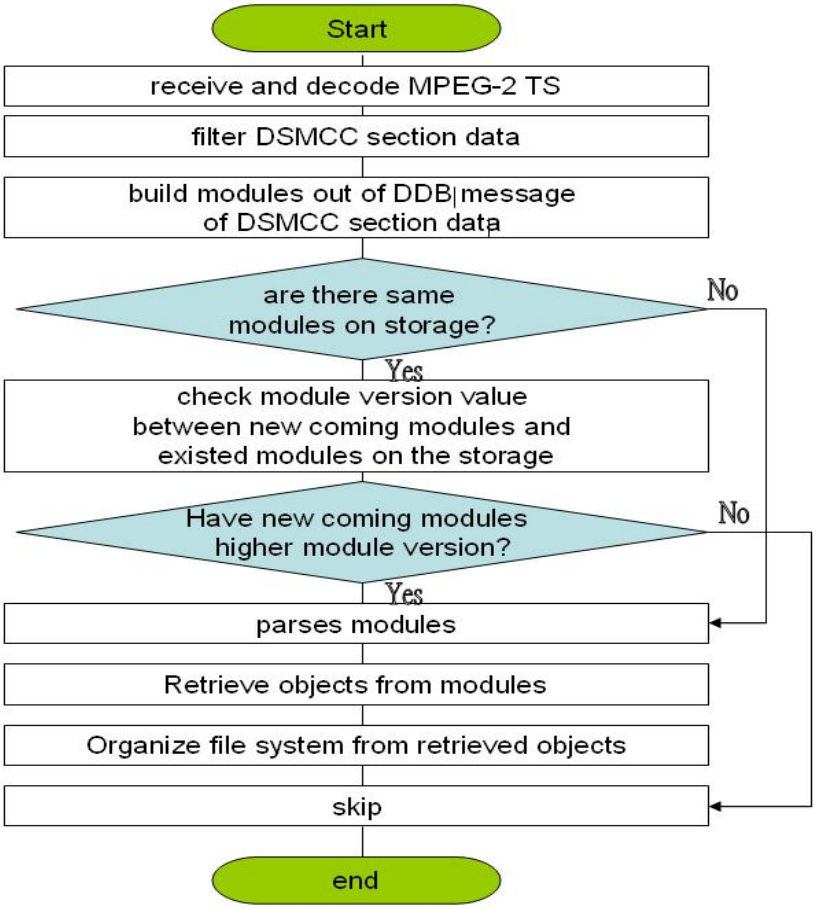

Figure 10. Flow chart of kwon's retrieving object

2.4. Real-Time Carousel Caching and Monitoring Architecture

當 Obecjt Carousel 在傳送時，它所包含的資料內容就已經無法被 Broadcaster 所更動，所以如果資料內容有所更新時，broadcaster 會重新發送這份資料，並且在 Version Number 的欄位上註明，所以 Version Manager 用來即時監控是不是有較新的版本，是不是要重新接收已經收過的資料。另外 Monitor Threads 則是用來監控各自所屬的執行緒目前的執行狀況，並且適時的通知下一個階段的執行緒開始處理資料。

Dong-Hwan Park等人針對Eun-Jung Kwon的系統加以改進，提出了Object Acquiring TimeSaving Scheme這種作法[7]，在STB端提出的一個對Object Carousel即時監控處理的作法，架構主要分成Object Cache及Xlet Cache兩個部

份。在Object Cache中的Real-Time Thread Pool中有用來監控目前的各種不同資料狀況的執行緒跟用來下載資料的執行緒兩種，像Download Info Indication (DII) 記錄著我們要下載的資料被分成了幾個modules，然後每個module中被分成幾個 blocks； Download Data Block (DDB) 中記錄著真實的資料內容，所以我們需要監控著DII是否已經下載完成。當DII's monitor thread發出DII已經下載完成的訊息之後，DSM-CC Parser才會開始接收處理DDB Cache中的資料，並且送到Xlet Cache中。而Xlet Cache主要就是控制xlet的執行跟來自不同channel的xlets之間互相切換的資料暫存。它的架構圖如Figure 11所示。

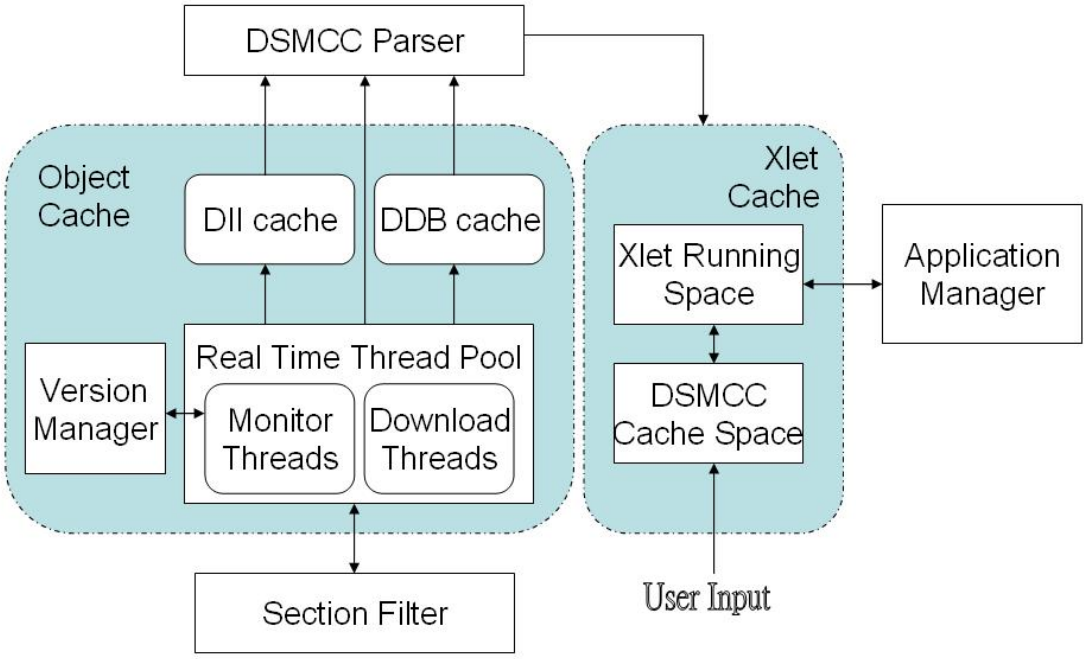

Figure 11. Overview of DSM-CC Object Carousel Architecture

2.5. Multimedia Information Broadcasting Ecoding and Decoding

要如何對於一份資料作encode並且在資料被接收端接受之後，使接收者能夠快速而便利的decode，各家的作法不一，而且各種作法也各有利弊。其中L.Atzori 等人對於web applications(DVB-HTML)提出了一套完整而簡易的encoding/decoding的作法[8]，這個設計實作簡易而且實用性高。

在encode的部份，爲了保持資料更新的即時性，L.Atzori把資料內容的index，要處理的webs數量，以及這些資料的根目錄都記錄在configuration file當中，以方便在每一次的encoding開始前可以修改encoding的資料內容。在讀取configuration file之後，就開始建立program association table(PAT)跟program map table(PMT)的table，再來就根據configuration file中所記錄資料的樹狀結構，從資料夾的結構開始encoding，最後再針對資料的內容作處理，整個encoding的流程如Figure 12所示。

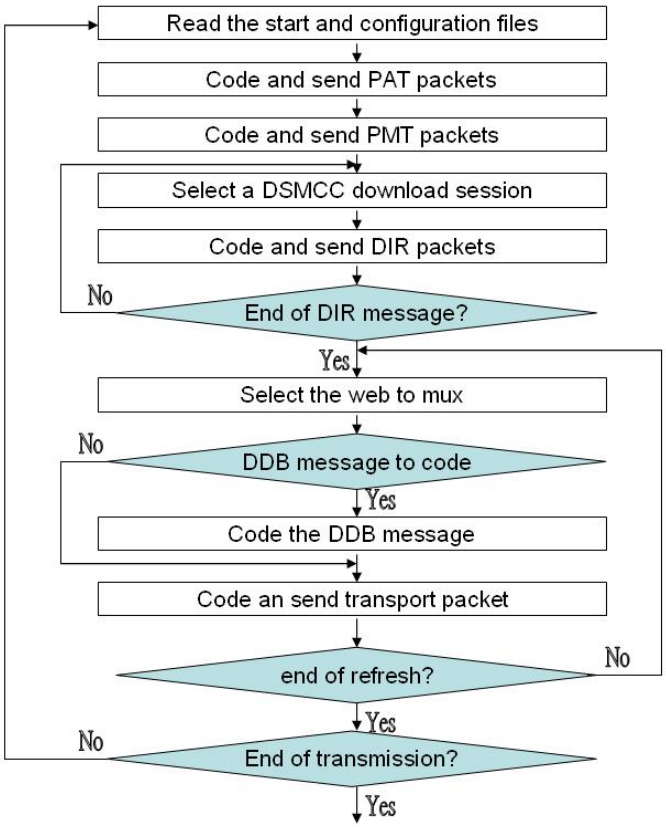

Figure 12. Atzori's encoding scheme

在接收者端，當收到送過來的 transport stream packets 之後，就依序開始處理 PAT、PMT 的資料。處理完 PMT 之後，就可以從 PMT 的內容得知哪些 packets 包含有我們需要的資料夾的數狀結構及資料內容，這時就可以將資料一一解碼並重建資料格式了，decoding 的流程圖如 Figure 14 所示。

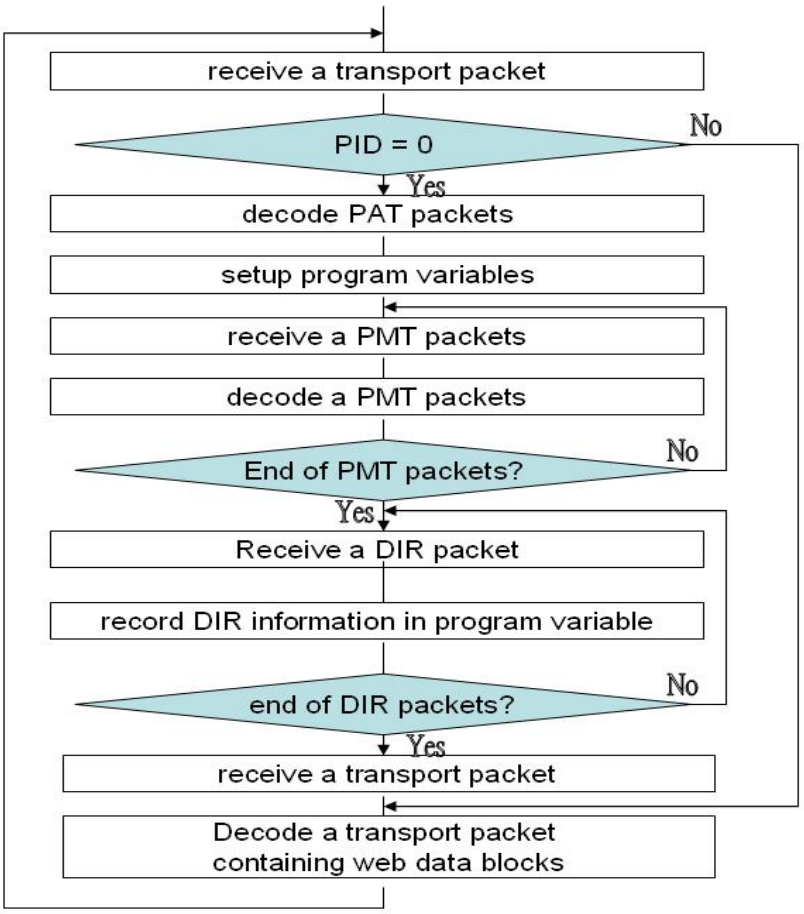

Figure 13. Atzori's dncoding scheme

2.6. Summary

綜合以上所述，我們的設計需求大概有以下幾點：

1) 參照 Hongguang Zhang 的想法，設計一個快速且有效率的 Section Filter。另外要能提供更有效率分辨每一份資料是不是已經重覆接收的機制。

2) 能夠快速處理 Object Carousel 並且有效率的判斷 version 是不是有更新，資料要不要重新接收。並且決定 DDB 的資料是不是該處理，或者是要整個放棄重新接收。

3. 系統設計概念與背景

本論文的主要目標是建置一個符合 MHP 且具有高效能的數位電視系統 DSM-CC 解碼與管理系統，並且降低整個處理過程的硬體資源 (如記憶體)使用，增加整體處理的速度。此外，爲了達到驗證 DSM-CC 系統之可行性與正確性之目標，當一個 transport stream 被接收到我們的平台之後，此實作系統必須能夠分析該 TS，取出 DSM-CC 相關的 packets，包括了給應用程式管理者使用的應用程式的資訊以及真正可以執行的應用程式的資料，解碼並重組之後，還原回原本的可執行物件以及資料格式。

在各種資料的規格書當中雖然清楚地訂定了規範格式，然而要處理的資料型態眾多，且資料內容複雜，如何有效率地識別各種資料型態並進行對應的處理則成爲影響整體 DSM-CC 執行效能的重要議題。

MHP 的應用程式中，我們主要著重在 DVB-J 的應用程式，也就是一般所謂的 xlet。但是，DVB-J 的應用程式並不是單純只有 xlet 而已，包括其它執行時有需要用到的資料，像是圖片(image/jpg、bmp)、聲音(audio/mp3、wmv)，檔案(file/txt、class)，等也都會和 xlet 一併被包裝成爲 Object Carousel(OC)的型態。

在MHP規格中, xlet必須被包裝爲Object Carousel(OC)的型態才能由傳送者端傳送到接收端（STB），並且爲了在網路上傳輸時能保持其效率及正確性，又被層層包裝爲Transport Stream。TS由傳送端透過broadcast送到接收者端，由接收者端解碼OC並還原爲xlet，才能在STB被執行。Figure 14應用程式、OC以及TS之間的關係圖。

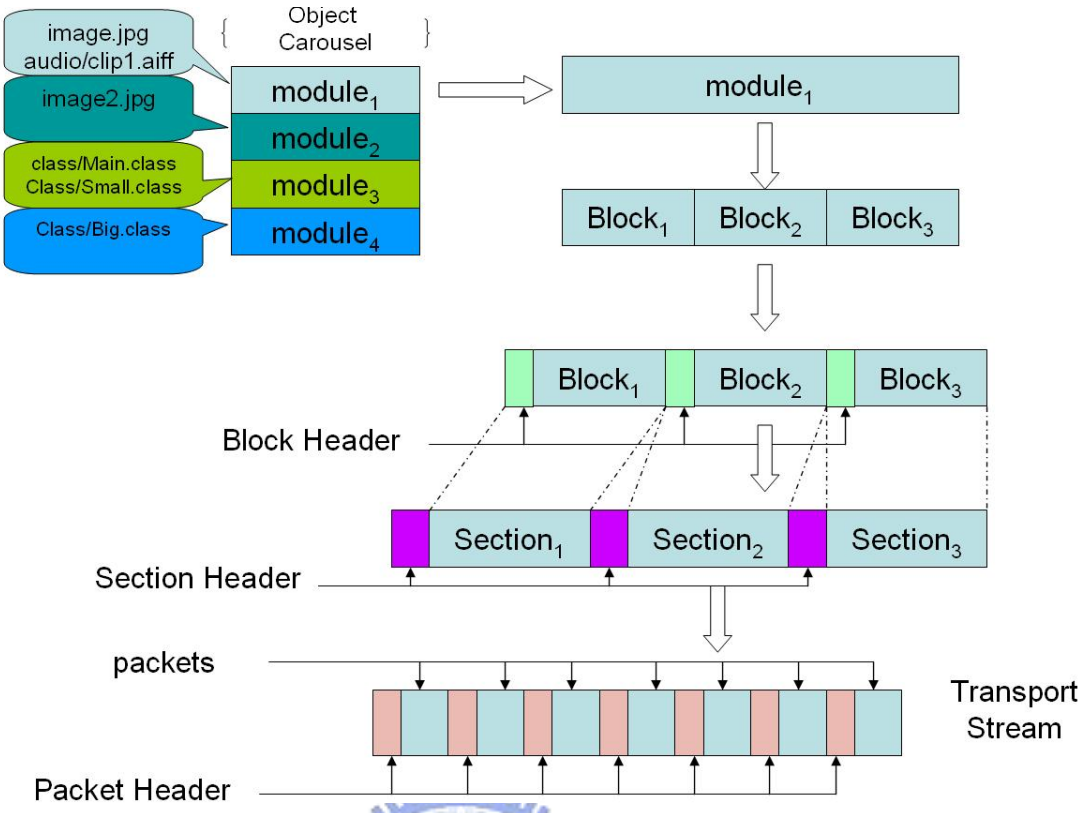

Figure 14. Relation Of applications, OC, modules, sections , and TS

可執行在使用者端的數位電視應用程式，通常包含執行檔(.class 檔)以及執行時所需的資料檔案(data：包括圖形檔，說明，聲音…等等.)。爲了表示以及保存這些檔案之間的關連性，因此通常以 OC 的型式包裝，而這些關聯性分別存放在 Program Association Table(PAT)與 Program Map Table(PMT)中。此外，爲了適應各種不同頻寬的網路，所以把這些 OC，PAT 以及 PMT 切成一個個大小固定的 packets。並以 packets 在網路上傳送，爲了正確取出這些具有關聯性的資料，因此需要知道這些資料所在的 packets ID (PID)。

從處理的流程來看，當broadcaster把TS送出來之後，由近端的接收者(tuner及CA)接收特定頻段的資料串流，並將資料串流送給Demux處理，而Demux會對於固定的長度 188 bytes作讀取(也就是一次讀取一個packet)。然後透過處理如 Figure 15中的TS Header的部份(TS Header的詳細處理將在後面描述)，把每一個packet的PID取出，並將PID = 0x0000 的packet(也就是PAT)送給SectionFilter重新組合之後，再傳給Service Information(SI)作解碼的工作，其中PID的對應表如Table

1所示。

Syntax	No. of bits
transport_packet(){	
sync_byte	**8**
transport_error_indicator	**1**
payload_unit_start_indicator	**1**
transport_priority	**1**
PID	**13**
transport_scrambling_control	**2**
adaptation_field_control	**2**
continuity_counter	**4**
if(adaptation_field_control==’10’|| adaptation_field_control==’11’){	
adaptation_field()	
}	
if(adaptation_field_control==’01’|| adaptation_field_control==’11’){	
for(i=0 ; i < N ; i++){	
data_byte	**8**
}	
}	
}	

Figure 15. Transport packet syntax

Table 1. PID Table

Value	description
0x0000	Program Association Table
0x0001	Conditional Access Table
0x0002-0x000F	reserved
0x0010-0x1FFE	may be assigned as network_PID, Program_map_PID, elementary_PID, or for other purposes
0x1FFF	Null packet

而 Service Information 處理過 PAT 之後，會得到 PMT 的相對應 PID，再把這些 PID 的值送回去給 Demux，以便從 transport stream 中取出 PMT 相對應的 packets 傳給 Section Filter 處理。

最後 Service Information 處理過 PMT 並且建立起 PID 跟 stream type 的 table 之後，我們就可以知道每一個 packet 是屬於哪一種類型的資料，並且從當中抽取出我們打算要處理的部份，經過 Cyclic Redundancy Check(CRC)確認無誤之後，送給相對應的 parser 作處理了，就本論文的的目標，則是依序送給 Section Filter 跟 DSM-CC 處理。

3.1. Transport Stream

在 transport stream 的格式中，每一個 pakcet 長都被固定爲 188 bytes，其中主要可以分成 Service Information 、 Program Specific Information (PSI)、Packetised Elementary Stream (PES)，這幾種。

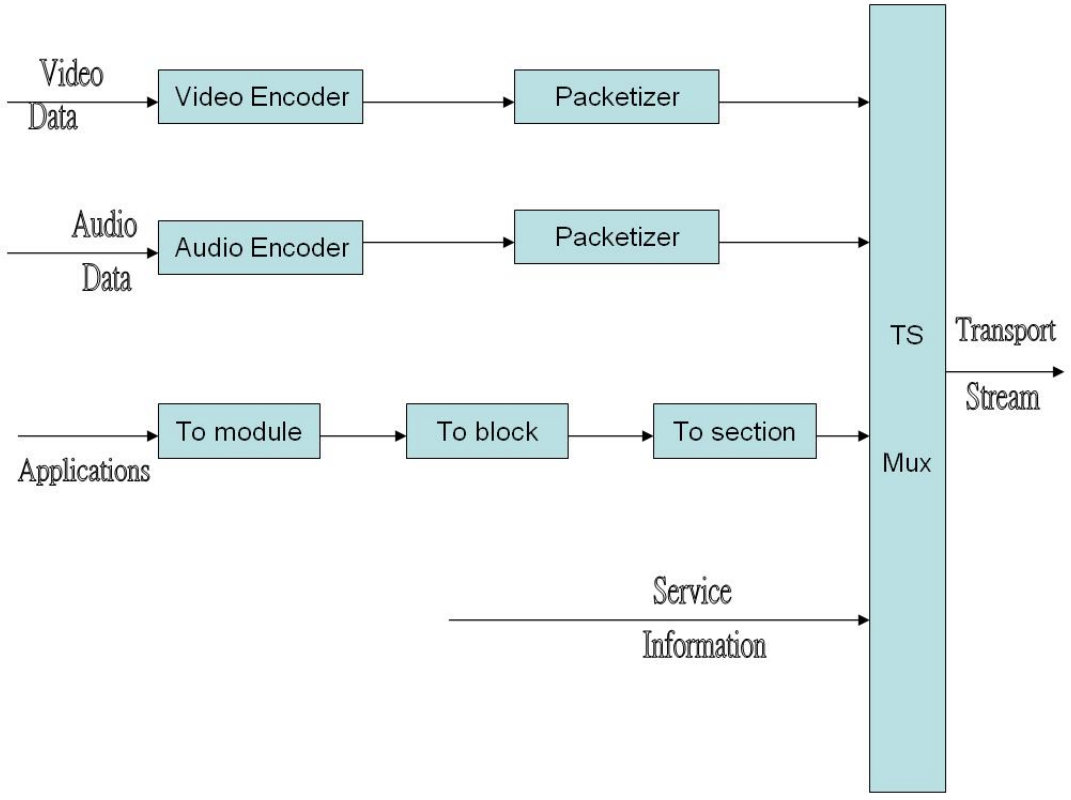

Figure 16. Data to Transport Stream

如Figure 16所示，Video跟Audio的資料被轉成PES。而一般的應用程式被模組化然後切成一個個的block，再轉成sections，最後再分割成 188 bytes的packets。

之後PES，PSI，Service Information的資料，再由TS Mux轉成transport stream。只是不論是PES、PSI或是SI，它們被轉成transport stream時，都會被加上類似的TS Header，而我們也可以從讀取TS Header的過程中，取得這個packet的PID，所以再來就先看看TS Header的格式。

3.1.1. Transport Stream Header

Transport Stream Header的格式如Figure 15中所示：

- sync_byte，長度爲 1 byte，這是用來確認一個 packet 的開頭，所以其值固定爲 0x47。不論是哪一種類型的資料，它們在 transport stream 的層級都是由 TS sync_byte 開始的。
- transport_error_indicator，長度爲 1 bit，如果被設爲'1'的話，表示這個 packet 在傳送的過程當中，曾經出現錯誤。
- payload_unit_start_indicator，長度爲 1 bit，可以分成兩個情況來討論，如果是 Program Specific Information (PSI)的話
  - 若被設爲'1'的話，表示這個 packet 中存在某一個 section 的開頭，則在 continuity_counter 之後，會多出 1 byte 的 pointer_field。
  - 若被設爲'0'的話，表示這個 packet 中所有的資料都是屬於同一個 section，則不會有 pointer_field 存在。

  如果是 Packetised Elementary Stream(PES)的話
  - 若被設爲'1'的話，表示這個 packet 是某一個 section 的開頭。
  - 若被設爲'0'的話，表示這個 packet 當中所有的資料都是屬於某一個 section 中的一部份。
- PID，長度爲 13 bits，透過 PID 我們可以得知這個 packet 的 payload 中所記錄的是哪一種類型的資料。

- pointer_field，長度爲 1 byte，用來指出在這個 packet 中，下一個 section 的開頭在哪個位置，如果 pointer_field 是'0'的話，表示整個 payload 都是屬於同一個 section，如果 pointer_field 不爲'0'的話，則它所表示的數字(n)是從 pointer_field 的下一個 byte 開始的第 n 個 byte，是一個新的 section 的開頭。

3.2. Service Information

在Service Information的工作當中，只有PAT以及PMT的部份與DSM-CC相關。PAT當中記錄了有關於PMT的PID；而PMT當中記錄了所有TS所含的資料，資料型態, 以及所對應的PID。如Figure 17所示。

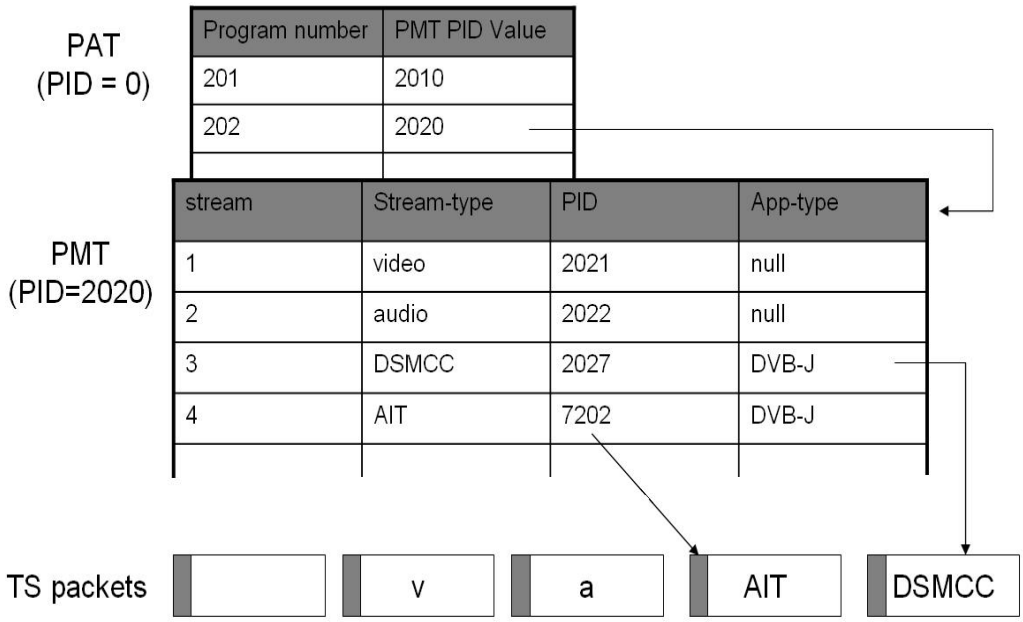

Figure 17. The procedure of PAT and PMT parsing

3.2.1. Program Association Table

PAT當中記錄著跟PMT相關的資訊，包括了program_number跟PID。傳送PAT的packet的PID固定爲 0x0000，這項訊息包含在傳送PAT的TS header中，所以SI可以輕易地判斷PAT所在的TS, 並從TS中將PAT取出來並送給PAT parser處理。

PAT的Syntax如Figure 18所示。

- table_id，長度爲 8 bits，在 PAT 它的值固定爲 0x00。
- section_length，長度爲 12 bits，記錄了這個 section 的總長度，我們可以從 section_length 這個欄位記錄的資料，確認 PAT 的真正長度，並且將後面用來填充至 188 bytes 的 null bytes 除去。
- program_number，長度是 16 bits，用來記錄每一個 PMT 的 program number。例如 Figure 6 當中的 201、202。
- program_map_PID：長度是 13 bits，記錄了每一個 PMT 的 PID。例如 Figure 6 當中的 2010、2020。

Syntax	**No. of bits**
program_association_section(){	
table_id	**8**
section_syntax_indicator	**1**
'0'	**1**
reserved	**2**
section_length	**12**
transport_stream_id	**16**
reserved	**2**
version_number	**5**
current_next_indicator	**1**
section_number	**8**
last_ section_number	**8**
for(i = 0 ; i < N ; i++){	
program_number	**16**
reserved	**3**
if(program_number=='0'){	
network_PID	**13**
}	
else{	
program_map_PID	**13**
}	
}	
CRC_32	**32**
}	

Figure 18. Program association Table

3.2.2. Program Map Table

當我們透過Service Information解出了PAT當中所記錄的PMT PID之後，便可以把這些資訊送回去給Demux處理，而Demux會把這些PID對應到的packet抽出依序組成完整的Program Map Table，並傳回去給Service Information處理。如Figure 19所示。

- table_id，長度爲 8 bits，在 PMT 中，其值固定爲 0x02。
- section_length，長度爲 12 bits，記錄了這個 section 的總長度，我們可以從 section_length 這個欄位記錄的資料，確認 PMT 的真正長度，並且將後面用來填充至 188 bytes 的 null bytes 除去。
- program_number，長度是 16 bits，記錄了這個 PMT 的 program number，可以和 PAT 中所記錄的 program number 作比對。
- stream_type，長度是 8 bits，記錄了這個elementary_PID所在的packet是屬於哪一種type，如Table 2所示。
- elementary_PID，長度 13 bits，記錄了各種 PID，讓 Demux 可以在適當的時機，透過 PID 將我們需要的 packet 抽出來重組成爲完整的 sections。

Syntax	**No. of bits**
TS_program_map_section(){	
table_id	**8**
section_syntax_indicator	**1**
‘0’	**1**
reserved	**2**
section_length	**12**
program_umber	**16**
reserved	**2**
version_number	**5**
current_next_indicator	**1**
section_number	**8**
last_ section_number	**8**
reserved	**3**
PCR_PID	**13**
Program_info_length	**12**
for(i = 0 ; i < N ; i++){	
descriptor()	
}	
for(i = 0 ; i < N1 ; i++){	
stream_type	**8**
reserved	**3**
elementary_PID	**13**
reserved	**4**
ES_info_length	**12**
for(i = 0 ; i < N2 ; i++){	
descriptor()	
}	
}	
CRC_32	**32**
}	

Figure 19. Program Map Table

Table 2. Stream Type

Value	Description
0x00	ITU-T | ISO/IEC Reserved
0x01	ISO/IEC 11172 Video
0x02	ITU-T Rec. H.262 | ISO/IEC 13818-2 Video or ISO/IEC 11172-2 constrained parameter video stream
0x03	ISO/IEC 11172 Audio
0x04	ISO/IEC 13818-3 Audio
0x05	ITU-T Rec. H.222.0 | ISO/IEC 13818-1 private_sections
0x06	ITU-T Rec. H.222.0 | ISO/IEC 13818-1 PES packets containing private data
0x07	ISO/IEC 13522 MHEG
0x08	ITU-T Rec. H.222.0 | ISO/IEC 13818-1 Annex A DSM CC
0x09	ITU-T Rec. H.222.1
0x0A	ISO/IEC 13818-6 type A
0x0B	ISO/IEC 13818-6 type B
0x0C	ISO/IEC 13818-6 type C
0x0D	ISO/IEC 13818-6 type D
0x0E	ISO/IEC 13818-1 auxiliary
0x0F-0x7F	ITU-T Rec. H.222.0 | ISO/IEC 13818-1 Reserved
0x80-0xFF	User Private

3.3. Application Information Table

當應用程式被下載之後，STB必須讓使用者得知目前有哪些應用程式可以執行，並且提供使用者關於這些應用程式的介紹。而這部份的資訊，則被包括在Application Information Table(AIT)當中[1][9][10]。

AIT包含的資訊包括應用程式名稱、類型、說明、圖像、以及來源位置。AIT是屬於應用程式管理者(Application Manager)的一部份。 因爲它記錄了跟應用程式相關的資訊，爲了驗證DSM-CC的完整性與正確性，本論文特別將它獨立出

來討論，並且提供介面給應用程式管理者在必要的時候去存取相關的資料。AIT 的格式如Figure 20所示。

- table_id，長度爲 8 bits，在 AIT 當中，其值固定爲 0x74。
- section_length，長度爲 12 bits，記錄了這個 section 的總長度，我們可以從 section_length 這個欄位記錄的資料，確認 AIT 的真正長度，並且將後面用來填充至 188 bytes 的 null bytes 除去。
- application_type，長度爲 15 bits，設定這個AIT中所記錄的應用程式的類型，主要有DVB-J跟DVN-HTML兩種，如Table 3所示。
- application_loop_length，長度爲 12 bits，記錄了這張 AIT 後面所儲存的資料總長度。
- application_identifier，可以分爲長 32 bits 的 organiztion_id 以及長 16 bits 的 application_id，而 organiztaion_id 是用來確認提供這個應用程式的公司，而 application_id 則是表示這個應用程式是這家公司提供的第幾個。此外，organization_id 是要透過 DVB 組織註冊的，每一家公司都有自己的 organization_id。
- application_control_code，長度是 8 bits，指出這個應用程式存到記憶體中的初始狀態，如Table 4所示。
- application_descriptors_loop_length，長度是 12 bits，記錄了後面所有descriptor的總長度，descriptor的主要種類，大致上如Table 5所示。

Syntax	**No. of bits**
application_information_section(){	
table_id	**8**
section_syntax_indicator	**1**
reserved_future_use	**1**
reserved	**2**
section_length	**12**
test_application_flag	**1**
application_type	**15**
reserved	**2**
version_number	**5**
current_next_indicator	**1**
section_number	**8**
last_ section_number	**8**
reserved_future_use	**4**
common_descriptors_length	**12**
for (i = 0 ; i < N ; i++){	
descriptor()	
}	
reserved_future_use	**4**
application_loop_length	**12**
for(i = 0 ; i < N ; i++){	
application_identifier()	
application_control_code	**8**
reserved_future_use	**4**
application_descriptors_loop_length	**12**
for(j = 0 ; j < N ; j++){	
descriptor()	
}	
}	
CRC_32	**32**
}	

Figure 20. Syntax of Application Information Table

Table 3. Applcation Type

Application_type	description
0x0000	reserved_future_used
0x0001	DVB-J application
0x0002	DVB-HTML application
0x0003-0x7FFF	subject to registration with DVB

Table 4. DVB-J application control code values

Code	Identifier
0x01	AUTOSTART
0x02	PRESENT
0x03	DESTORY
0x04	KILL
0x05	reserved_future_used
0x06	REMOTE

Table 5. Descriptors and their location in AIT

Descripotr	Mandatory	Location
application descriptor	mandatory	application descriptor loop
application name descriptor	mandatory	application descriptor loop
Application icons descriptor	optional	application descriptor loop
DVB-J application descriptor	mandatory for DVB-J application	application descriptor loop
DVB-J application location descriptor	mandatory for DVB-J application	application descriptor loop

3.3.1. Application Descriptor

在數位電視的環境底下，一個 service 指的是由 Service Information，video stream(s)，audio stream(s)，及應用程式所形成，彼此之間具有關聯性的單一集合。例如：一個電視頻道所提供的完整 stream 就可以視爲一個 service。

而Application Descriptor提供了關於這個應用程式的一些必要的基本訊息給接收者。一些訊息的例子如下：該應用程式是不是跟service綁在一起、以及這個應用程式能不能讓使用者直接看到，或是只能給其它應用程式讀取。而application descritpor會在每一個AIT entry中出現，且每一個應用程式會有正好一個

application descriptor。它的格式如Figure 21所示。

- descriptor_tag，長度爲 8 bits，值固定爲 0x00，用來確認被記錄的是 application descriptor。
- application_profiles_length，長度爲 8 bits，用來記錄 application_profile loop 的長度。
- service_bound_flag，長度爲 1 bit，用來指明這個應用程式是不是和這個 service 綁在一起。如果是的話，當使用者選擇切換到另外一個 service 的同時，這個應用程式會被移除。
- visibility，長度爲 2 bits，用來告訴接收者哪些應用程式是使用者可視，哪些應用程式是只能讓其它應用程式看到，哪些是兩者都可以看到的。如Table 6所示。

Syntax	**No. of bits**
application_descriptor(){	
descriptor_tag	**8**
descriptor_length	**8**
application_profiles_length	**8**
for(i=0; i<N; i++) {	
application_profile	**16**
version.major	**8**
version.minor	**8**
version.micro	**8**
}	
service_bound_flag	**1**
visibility	**2**
reserved_future_use	**5**
application_priority	**8**
for(i=0; i<N; i++) {	
transport_protocol_label	**8**
}	
}	

Figure 21. Syntax of Application Descriptor

Table 6. Definition of visibility states for applications

visibility	description
00	This application shall not be visible either to applications via an application listing API or to users via the navigator with the exception pf any error reporting or logging facility, etc.
01	This application shall not be visible to users but shall be visible to applications via an application listing API.
10	reserved_future_use
11	This application can be visible to users and shall be visible to applications via the application listing API.

3.3.2. Application Name Descriptor

Application name descriptor記錄了這一個應用程式的名稱，應用程式管理者會把這個名稱顯示給一般使用者看，它的格式如Figure 22所示。

- descriptor_tag，長度爲 8 bits，值固定爲 0x01，用來確認被記錄的是 application name descriptor。
- ISO_639_language_code，長度爲 24 bits，使用了 ISO 639.2 three character language code，用來指明這個應用程式的名稱是用哪一種語言記錄的。
- application_name_length，長度爲 8 bits，用來記錄後面用了幾個 bytes 來儲存這個應用程式的名字。
- application_name_char，記錄了這個應用程式要顯示給使用者的名稱。

Syntax	**No. of bits**
application_name_descriptor(){	
descriptor_tag	8
descriptor_length	8
for (i=0; i<N; i++){	
ISO_639_language_code	24
application_name_length	8
for (i=0; i<N; i++){	
application_name_char	8
}	
}	
}	

Figure 22. Syntax of the application name descriptor

3.3.3. Application Icons Descriptor

Application icons descriptor在AIT當中屬於選擇性的參數，會記錄影像或是可用來表示這一個應用程式的圖片。它的格式如Figure 23所示。

- descriptor_tag，長度爲 8 bits，值固定爲 0x0B，用來確認被記錄的是 application icons descriptor。
- descriptor_length，長度爲 8 bits，用來記錄這個 descriptor 的長度。
- icon_locator_length，長度爲 8 bits，用來記錄 icon 所在的檔案名稱長度。
- icon_locator_byte，具體指明了 icon 的真正位置。

Syntax	**No. of bits**
application_icons_descriptor(){	
descriptor_tag	**8**
descriptor_length	**8**
icon_locator_length	**8**
for (i=0; i<N; i++){	
icon_locator_byte	**8**
}	
icon_flags	**16**
for (i=0; i<N; i++){	
reserved_future_use	**8**
}	
}	

Figure 23. Syntax of Application Icons Descriptor

3.3.4. DVB-J Application Descriptor

DVB-J Application Descriptor告訴接收者當這個應用程式要被執行時，需要傳遞哪些參數給這個應用程式。它的格式如Figure 24所示。

- descriptor_tag，長度爲 8 bits，值固定爲 0x03，用來確認被記錄的是 DVB-J application descriptor。
- parameter_length，長度爲 8 bits，記錄了要傳遞的參數的長度。
- parameter_byte，這是一個 string 的陣列，記錄了所有要傳給這個應用程式的參數。

Syntax	No. of bits
dvb_j_application_descriptor(){	
descriptor_tag	**8**
descriptor_length	**8**
for(i=0; i<N; i++){	
parameter_length	**8**
for(j=0; j<parameter_length; j++){	
parameter_byte	**8**
}	
}	
}	

Figure 24. Syntax of DVB-J Application Descriptor

3.3.5. DVB-J Application Location Descriptor

DVB-J application location descriptor用來告訴接收者這個應用程式的base directory在哪邊，標示應用程式的main class，以及指示與應用程式相關的class時的路徑。它的格式如Figure 25所示。

- descriptor_tag，長度爲 8 bits，值固定爲 0x04，用來確認被記錄的是 DVB-J application location descriptor。
- base_directory_length，長度爲 8 bits，用來指明 base_directory_byte 的長度。
- base_directory_byte，記錄了一個從 file system 的 root 開始的 directory name，而這個路徑也是這個應用程式要開始執行的 base path name。
- classpath_extension_byte，記錄了除了 base directory 之外，如果還要搜查其他的 classpath 的話，這些 classpath 被存放的路徑。
- initial_class_byte，包含了一個 string，記錄了 the name of the object in the system that is the class implementing the Xlet interface。而這個 string 是一

個 DVB-J 可以在 classpath 中找到的 class name，例：”com.broadcaster.appA.MainClass”。

Syntax	**No. of bits**
dvb_j_application_location_descriptor{	
descriptor_tag	**8**
descriptor_length	**8**
base_directory_length	**8**
for(i=0; i<N; i++){	
base_directory_byte	**8**
}	
Classpath_extension_length	**8**
for(i=0; i<N; i++){	
classpath_extension_byte	**8**
}	
for(i=0; i<N; i++){	
initial_class_byte	**8**
}	
}	

Figure 25. Syntax of DVB-J Application Location Descriptor

3.4. Section Filter

在經過 Demux 以及 Service Information 的處理之後，SI 擁有 PMT，可得知每一個 PID 所對應到的 packet 是屬於什麼樣的類型的應用程式，而 DSM-CC 這時就可以經由 SI 取得 DSM-CC 相關資料的 PID。DSM-CC 再以所欲取得之資料的 PID 向 Section Filter 取得包含 DSM-CC 資料的 sections。

在Figure 26中，我們可以看出sections，blocks，modules，還有DSM-CC object carousel之間的關係。當SF收到各類資料的sections，會把從Demux端接收到的sections按照不同的類型送給對應的功能模組處理；當SF收到DSM-CC相關的sections，則將其交給DSM-CC功能模組處理。

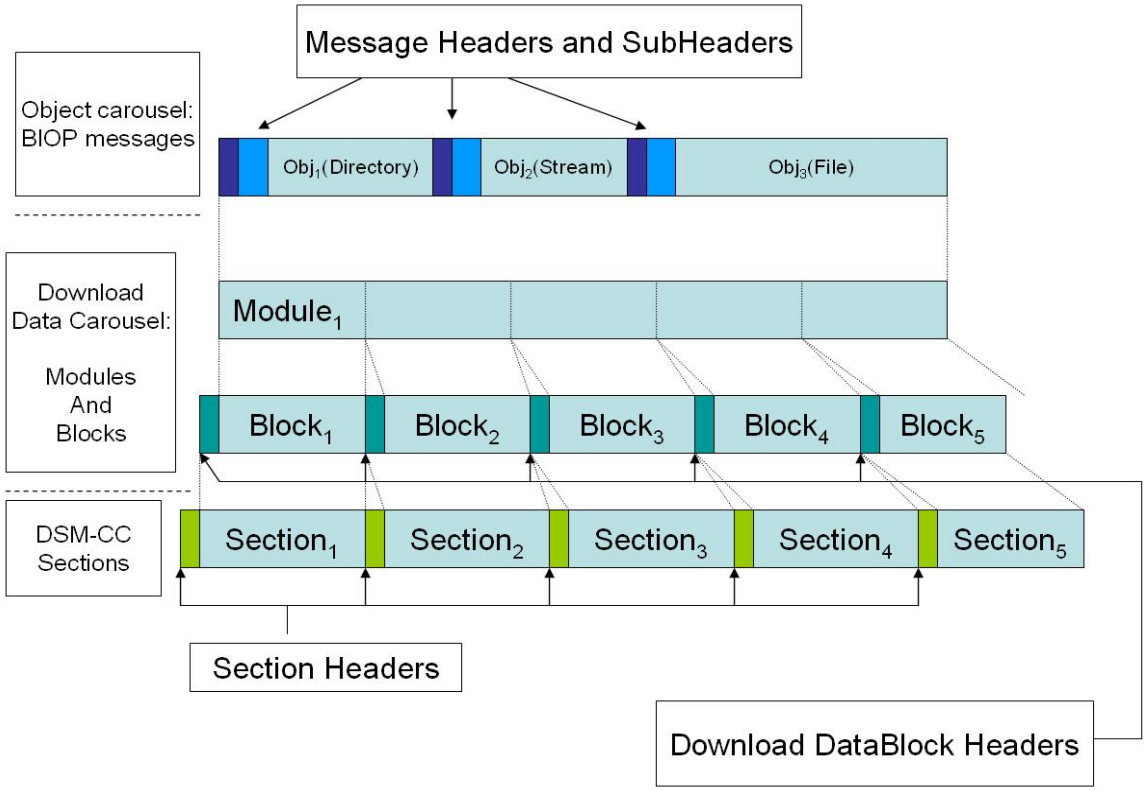

Figure 26. Relationship between sections, blocks, modules, and DSM-CC object carousel

3.4.1. Sections

從Demux端收到的sections是由一個或是多個packets所組合而成的，而其中每個section的長度最大不超過 4099 bytes，section的syntax如Figure 27所示。

- table_id，長度是 8 bits，我們主要用到的table_id值爲 0x3B跟 0x3C兩種，0x3B 對 應 到 的 是 Download Server Initiate(DSI) 跟 Download Info Indication(DII)兩種，0x3C對應到的是Download Data Block(DDB)，table_id的值在DSM-CC中的定義如Table 7所示。
- dsmcc_section_length，長度爲 12 bits，用來記錄這個 section 的真正長度，所以一個 section 的長度最多不超過 4099bytes。
- table_id_extension，長度爲 16 bits，只有在 table_id 是 0x3B 時有用，功能是判別這個 section 是屬於 DSI 或是 DII。如果 table_id_extension 的

值是 0x0000 及 0x0001 的話，那這是屬於 DSI，其他 table_id_extension 的值都是屬於 DII。

Syntax	**No. of bits**
DSMCC_section(){	
table_id	**8**
section_syntax_indicator	**1**
private_indicator	**1**
reserved	**2**
dsmcc_section_length	**12**
table_id_extension	**16**
reserved	**2**
version_number	**5**
current_next_indicator	**1**
section_number	**8**
last_section_number	**8**
if(table_id == 0x3A){	
LLCSNAP()	
}	
else if (table_id == 0x3B){	
userNetworkMessage()	
}	
else if (table_id == 0x3C){	
downloadDataMessage()	
}	
else if (table_id == 0x3D){	
DSMCC_descriptor_list()	
}	
else if (table_id == 0x3E){	
for (i = 0;i < dsmcc_section_length - 9;i++){	
private_data_byte	
}	
}	
if(section_syntax_indicator == "0"){	
checksum	**32**
}	
else {	
CRC_32	**32**
}	
}	

Figure 27. Syntax of the DSM-CC section

Table 7. DSM-CC table_id assignment

Table_id	DSM-CC Section Type
0x3A	DSM-CC Section containing multi-protocol encapsulated data
0x3B	DSM-CC Section containing U-N Message, except Download Data Message
0x3C	DSM-CC Section containing Download Data Message
0x3D	DSM-CC Section containing Stream Descriptor
0x3E	DSM-CC Section containing private Data
0x3F	ISO/IEC 13818-6 reserved

3.5. Digital Storage Media Command and Control

DVB-MHP的規範是利用DSM-CC來傳送加值應用程式及其所需的資料的[11][12][13]，當機上盒系統收到了DSM-CC sections，可由section header判斷該section所含之資料的類型，再分別將這些sections依序排成完整的DSM-CC資料。當DSM-CC資料被重組完成之後，就可以傳給解碼器處理。解碼時，DSM-CC再分別以對應的解碼方式處理。DSM-CC共包含三種類型的資料：包含實際OC內容的DDB, 描述OC架構的DII, 以及帶有gateway參照資料的DSI。

3.5.1. DSI and DII

DSI跟DII用於描述資料來源之gateway以及OC內容的檔案層級架構，包含一個OC被分割成多少模組，module的id，size，模組共包含多少的DDB等等訊息，而這兩種格式的Header是一樣的，如Figure 28所示。

3.5.1.1. DSM-CC Message Header

- protocolDiscriminator，長度爲 1 bytee，值固定爲 0x11。
- DSM-CCType，長度爲 1 byte，主要是用來確認它是屬於MPEG-2 DSM-CC message中的哪一種type，它的值是 0x03。DSM-CCType的內容如Table 8所示。
- messageId，長度爲 2 bytes，用來確認要處理的message是屬於哪一種類

型，它的值由DSM-CCType來決定，值的對應表如Table 9所示。

- messageLength，長度爲 2 bytes，記錄了這個 message 真正的長度(包含 Header)。當我們要抓資料時，只要從 messageLength 扣掉 headerLength 就可以知道我們真正要抓取的資料長度了。

Syntax	**No. of bytes**
dsmccMessageHeader(){	
protocolDiscriminator	**1**
dsmccType	**1**
messageId	**2**
transactionId	**4**
reserved	**1**
adaptationLength	**1**
messageLength	**2**
if(adaptationLength > 0){	
dsmccAdaptationHeader()	
}	
}	

Figure 28. Syntax of Mpeg-2 DSM-CC Message Header

Table 8. MPEG-2 DSM-CC dsmccType values

dsmccType	**Description**
0x00	ISO/IEC 13818-6 Reserved
0x01	Identifies the message as an ISO/IEC 13818-6 IS User-to-Network configuration message.
0x02	Identifies the message as an ISO/IEC 13818-6 IS User-to-Network session message.
0x03	Identifies the message as an ISO/IEC 13818-6 IS Download message.
0x04	Identifies the message as an ISO/IEC 13818-6 IS SDB Channel Change Protocol message.
0x05	Identifies the message as an ISO/IEC 13818-6 IS User-to- Network pass-thru message.
0x06-0x7F	ISO/IEC 13818-6 Reserved.
0x80-0xFF	User Defined message type.

Table 9. DSM-CC Download messageId assignment

Message Name	messageId	Description
DownloadInfoRequest	0x1001	Client requests download parameters
DownloadInfoResponse, DownloadInfoIndication	0x1002	Download Server provides download parameters
DownloadDataBlock	0x1003	Download Server sends one download data block
DownloadDataRequest	0x1004	Client acknowledges downloaded data blocks
DownloadCancel	0x1005	Client or Download Server aborts the download scenario in progress
DownloadServerInitiate	0x1006	Download Server requests Client to initiate a download

3.5.1.2. DownloadServerInitiate Message

當應用程式極大, 以至於存在雙層的Object carousel時，DSI可用來記錄各個不同的DII之間的關係的。DSI的格式如Figure 29所示。

Syntax	**No. of bytes**
DownloadServerInitiate(){	
dsmccMessageHeader()	
serverId	**20**
compatibilityDescriptor()	
privateDateLength	**2**
for(i = 0 ; i < privateDateLength ; i++){	
privateDateByte	**1**
}	
}	

Figure 29. Syntax of DownloadServerInitiate Message

3.5.1.3. DownloadInfoIndication Message

DII記錄著跟Object carousel中包含的資料，包括了number of modules，module ID，module size…等等和我們要解出DDB中包含的資料時相關的訊息。DII的格式如Figure 30所示。

- downloadId，長度爲 4 bytes，用來確認我們所收到的每一個 block 都是屬於同一個 object carousel，downloadId 的值是在傳送時，由 network

指定，在之後使用的 DownloadDataBlock 都會有相同的 downloadId。

- blockSize，長度是 2 bytes，用來告知處理器這個 service 的所有 block 的大小，除了方便處理器對每一個 block 作處理之外，也可以讓我們方便得知每一個 module 被切成幾個 blocks。
- numberOfModules，長度是 2 bytes，用來記錄這個 service 中總共包含了幾個 modules。
- moduleId，長度 2 bytes，功用是用來確認 DownloadDataBlock 是屬於這個 module。
- moduleSize，長度是 4 bytes，用來記錄這個 module 的大小，配合前面的 blockSize 的話，我們可以得知這個 module 總共被切成幾個 blocks，並由此判斷我們是不是已經收到全部的 blocks。

Syntax	**No. of bit**
DownloadInfoIndication(){	
dsmccMessageHeader()	
downloadId	**4**
blockSize	**2**
windowSize	**1**
ackPeriod	**1**
tCDownloadWindow	**4**
tCDownloadScenario	**4**
compatibilityDescriptor()	
numberOfModules	**2**
for(i = 0;i < numberOfModules;i++){	
moduleId	**2**
moduleSize	**4**
moduleVersion	**1**
moduleInfoLength	**1**
for(i = 0;i < moduleInfoLength;i++){	
moduleInfoByte	**1**
}	
}	
privateDataLength	**2**
for(i = 0;i < privateDataLength;i++){	
privateDataByte	**1**
}	
}	

Figure 30. Syntax of DownloadInfoIndication Message

3.5.2. DownloadDataBlock Message

順利解出DSI跟DII當中包含的訊息之後，即可確認這一個service當中，總共包含了幾個modules，然後每一個module各自被切成幾個blocks，這時我們就可以一一的把真正存有資料的DDB抓進來，依序填到它們應該在的module中正確的位置，直到每一個module都被重新合併完成之後，再進行OC（以Broadcast Inter-ORB(Object Request Broker) Protocol(BIOP)格式編碼）的解碼。其中DDB的

格式如Figure 31，Figure 32所示。。

- downloadId，長度爲 4 bytes，用來和 DII 中的 downloadId 比對，以確定我們收到的是屬於同一個 service。
- moduleId，長度爲 2 bytes，用來確認這個 block 是屬於哪一個 module。
- blockNumber，長度爲 2 bytes，因爲每一個 module 都會被分成一個或是多個 blocks，但是在接收時，並沒有辦法保證我們收到跟處理是照次序的，所以要靠 blockNumber 幫它們重新照次序組合。

Syntax	**No. of bytes**
dsmccDownloadDataHeader(){	
protocolDiscriminator	**1**
dsmccType	**1**
messageId	**2**
tDownloadId	**4**
reserved	**1**
adaptationLength	**1**
messageLength	**2**
if(adaptationLength > 0){	
dsmccAdaptationHeader()	
}	
}	

Figure 31. Syntax of DownloadDataBlock Header

Syntax	**No. of bytes**
DownloadDataBlock(){	
dsmccDownloadDataHeader()	
moduleId	**2**
moduleVersion	**1**
reserved	**1**
blockNumber	**2**
for(i = 0 ; i < N ; i++){	
blockDataByte	**1**
}	
}	

Figure 32. Syntax of DownloadDataBlock

3.5.3. BIOP

BIOP 的全名爲 Broadcast Inter-ORB Protocol，它主要是由多個 modules 所組成的，裡面包含的就是最原始檔案的資料，包括了檔案名稱、檔案內容、檔案之間的結構。其中還可以再細分成爲 SrgandDir Message，File Message，Stream Message，StreamEvent Message 等種類。

3.5.3.1. BIOP SRGandDIR Message

當我們收到的一份完整的資料，包含資料夾、檔案，它們在原始的資料提供者端被以樹狀結構(tree structure)階層式儲存，而其中所有樹狀結構的根部節點(root)就是SrgandDir Message，也就是Service Gateway Message(Srg)，在一個OC中只會出現一次。之後再收到的每一個SrgandDir message都是屬於Directory message(Dir)，除非這份資料當中存在兩個以上的樹狀結構。SrgandDir message的格式如Figure 33所示。

- Magic，長度爲 4 bytes，它的值被固定爲 0x42494F50，用來確認這是 BIOP message 的開頭。
- objectKey_length，長度爲 1 byte，用來決定 objectKey_data_byte 的長度。
- objectKey_data_byte，長度由 objectKey_length 決定，如果是 Directory Message 的話，可以和 BIOP Profile Body 中的 objectKey_data_byte 比較來判斷這一個 Directory 的父節點是哪一個。如果是 File Message 的話，則可以用來判斷它所儲存的資料是屬於哪一個檔案。
- objectKind_length，長度爲 4 bytes，如果是 SRG Message 的話，其值爲 0x73726700，如果是 DIR Message 的話，其值爲 0x64697200。除了 objectKind_length 之外，SRG 跟 DIR 的其他格式都是一樣的。
- bindings_count，長度爲 2 bytes，主要是記錄這個資料夾底下，包含了幾個資料夾或是檔案。

- nameComponents_count，長度爲 1 byte，記錄了現在我們讀到的這份資料，它的名稱我們用幾個 bytes 來儲存它。
- id_length，記錄了我們現在讀到的這份資料，它名稱的長度。
- id_data_byte，真正記錄我們現在讀到的這份資料的名稱。
- kind_data_byte，記錄了我們現在讀到的這份資料是屬於哪一種類型，dir、fil、str，ste。
- IOR，如Figure 34所示。其中從IOP::taggedProfile()這個function之後的內容屬於BIOP Profile Body，如Figure 35所示。
- Profile_tag，長度爲 4 bytes，其值是 0x49534F06，用來確認這是 BIOP Profile Body 的開頭。
- objectKey_length，長度爲 1 byte，用來決定後面的 objectKey_data_byte 的長度，因爲它的名稱和之前在 BIOP 的 message 中出現過，所以這邊稱爲 profile_objectKey_length。
- objectKey_data_byte，長度由 profile_objectKey_length 決定，只有在 SrgandDir message 中會出現，用來和前面的 objectKey_data_byte 互相參考，以重建整個樹狀結構。

Syntax	No. of bits
BIOP::DirectoryMessage(){	
Magic	**4x8**
biop_version.major	**8**
biop_version.minor	**8**
byte_order	**8**
message_type	**8**
message_size	**32**
objectKey_length	**8**
for (i = 0; i < N1; i++){	
objectKey_data_byte	**8**
}	
objectKind_length	**32**
objectKind_data	**4x8**
objectInfo_length	**16**
for (i = 0; i < N2; i++){	
objectInfo_data_byte	**8**
}	
serviceContextList_count	**8**
for (i = 0; i < N3; i++){	
serviceContextList_data_byte	**8**
}	
messageBody_length	**32**
bindings_count	**16**
for (i = 0; i < N4; i++){	
BIOP::Name(){	
nameComponents_count	**8**
for (i = 0; i < N5; i++){	
id_length	**8**
for (j = 0; j < N6; j++){	
id_data_byte	**8**
}	
kind_length	**8**
for (j = 0; j < N7; j++){	
kind_data_byte	**8**
}	
}	
}	
bindingType	**8**
IOP::IOR()	
objectInfo_length	**16**
for (j = 0; j < N8; j++){	
objectInfo_data_byte	**8**
}	
}	
}	

Figure 33. Syntax of BIOP Directory Message

Syntax	No. of bits
IOP::IOR{	
type_id_length	**32**
for (i = 0; i < N1; i++){	
type_id_byte	**8**
}	
if (N1 % 4 != 0){	
for (i= 0; i < (4 - (N1 % 4)); i++){	
alignment_gap	**8**
}	
}	
taggedProfiles_count	**32**
for (n = 0; n < N2; n++){	
IOP::taggedProfile(){	
profileId_tag	**32**
profile_data_length	**32**
for (i = 0; i < N3; i++) {	
profile_data_byte	**8**
}	
}	
}	
}	

Figure 34. Syntax of IOP:IOR()

Syntax	**No. of bits**
BIOPProfileBody{	
profileId_tag	**32**
profile_data_length	**32**
profile_data_byte_order	**8**
liteComponents_count	**8**
BIOP::ObjectLocation{	
componentId_tag	**32**
component_data_length	**8**
carouselId	**32**
moduleId	**16**
version.major	**8**
version.minor	**8**
objectKey_length	**8**
for (k = 0; k < N2; k++){	
objectKey_data_byte	**8**
}	
}	
DSM::ConnBinder{	
componentId_tag	**32**
component_data_length	**8**
taps_count	**8**
BIOP::Tap{	
id	**16**
use	**16**
association_tag	**16**
selector_length	**8**
selector_type	**16**
transactionId	**32**
timeout	**32**
}	
for (m = 0; m < N3 - 1; m++){	
BIOP::Tap{	
id	**16**
use	**16**
association_tag	**16**
selector_length	**8**
for (i = 0; i < N4; i++){	
selector_data_byte	**8**
}	
}	
}	
}	
for (n = 0; n < N5;n++){	
BIOP::LiteComponent{	
componentId_tag	**32**
component_data_length	**8**
for (i = 0; i < N6; i++){	
component_data_byte	**8**
}	
}	
}	
}	

Figure 35. Syntax of BIOP Profile Body

3.5.3.2. BIOP File Message

BIOP file message當中，記錄了每一個對應到的檔名，它的真正檔案內容，格式如Figure 36所示。

- objectKind_data，長度爲 4 bytes，其值被固定爲 0x66696C00。
- content_data_byte，記錄了真正的檔案資料內容。

Syntax	**No. of bits**
BIOP::FileMessage(){	
Magic	**4x8**
biop_version.major	**8**
biop_version.minor	**8**
byte_order	**8**
message_type	**8**
message_size	**32**
objectKey_length	**8**
for (i = 0; i < N1; i++){	
objectKey_data_byte	**8**
}	
objectKind_length	**32**
objectKind_data	**4x8**
objectInfo_length	**16**
DSM::File::ContentSize	**64**
for (i = 0; i < N2 - 8; i++){	
objectInfo_data_byte	**8**
}	
serviceContextList_count	**8**
for (i = 0; i < N3; i++) {	
serviceContextList_data_byte	**8**
}	
messageBody_length	**32**
content_length	**32**
for (i = 0; i < N4; i++){	
content_data_byte	**8**
}	
}	

Figure 36. Syntax of BIOP File Message

3.5.3.3. Example

如Figure 37所示，這是一個BIOP資料原始樹狀結構的例子。下面的步驟就是我們程式讀取資料的過程。

1) 讀取到 dir，objectKey 是 06，然後它的 bindings_count 爲 1。

- fileName：$file_6$，profile_objectKey是 07。

2) 讀到 dir，objectKey 是 04，然後它的 bindings_count 爲 2。

- fileName：$file_3$，profile_objectKey是 05。
- dirName：dir_e，profile_objectKey是 06。

3) 讀到 dir，objectKey 是 08，然後它的 bindings_count 爲 1。

- fileName：$file_4$，profile_objectKey是 09。

4) 讀到 dir，objectKey 是 0A，然後它的 bindings_count 爲 1。

- fileName：$file_5$，profile_objectKey是 0B。

5) 讀到 srg，objectKey 是 01，然後它的 bindings_count 爲 5。

- fileName：$file_1$，profile_objectKey是 02。
- fileName：$file_2$，profile_objectKey是 03。
- dirName：dir_b，profile_objectKey是 04。
- dirName：dir_c，profile_objectKey是 08。
- dirName：dir_d，profile_objectKey是 0A。

6) 重建樹狀結構。

- 由srg開始，它下面包含了五個節點，分別是$file_1$，$file_2$，dir_b，dir_c，dir_e。
- dir_b的profile_objectKey爲 04，代表在step. 2 讀到具有objectKey = 04 的dir就是dir_b。
- dir_c的profile_objectKey爲 08，代表在step. 3 讀到具有objectKey = 08 的dir就是dir_c。

- dir_d的profile_objectKey爲 0A，代表在step. 4 讀到具有objectKey = 0A的dir就是dir_d。
- dir_e的profile_objectKey爲 06，代表在step. 1 讀到具有objectKey = 06的dir就是dir_e。

7) 讀到fil，objectKey是 02，和step. 5 中$file_1$的profile_objectKey相同，代表它儲存的是$file_1$的資料。

8) 讀到fil，objectKey是 03，和step. 5 中$file_2$的profile_objectKey相同，代表它儲存的是$file_2$的資料。

9) 讀到fil，objectKey是 05，和step. 2 中$file_3$的profile_objectKey相同，代表它儲存的是$file_3$的資料。

10) 讀到fil，objectKey是 07，和step. 1 中$file_6$的profile_objectKey相同，代表它儲存的是$file_6$的資料。

11) 讀到fil，objectKey是 09，和step. 3 中$file_4$的profile_objectKey相同，代表它儲存的是$file_4$的資料。

12) 讀到fil，objectKey是 0B，和step. 4 中$file_5$的profile_objectKey相同，代表它儲存的是$file_5$的資料。

處理到這邊，我們就可以把資料原本的被傳送端處理過之前的樣子，在接收者端重新建立起來，並且準備等 application manager 來讀取並執行了。

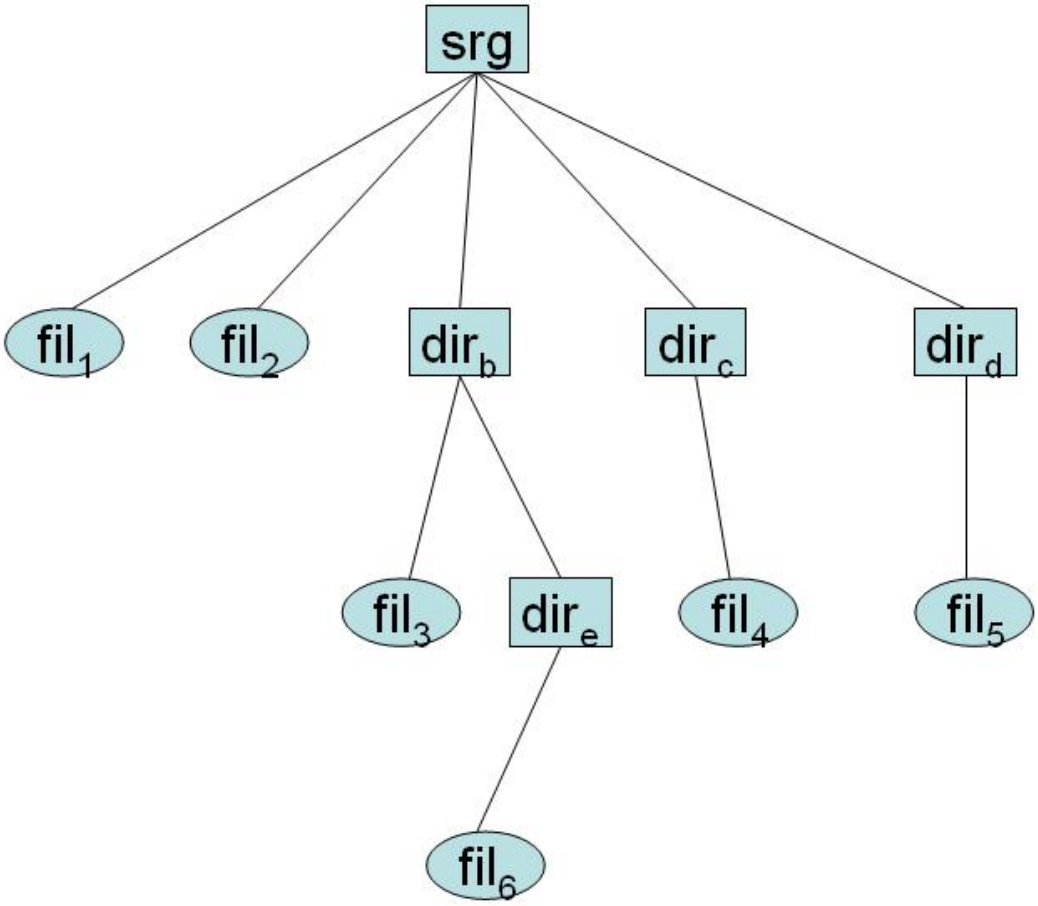

Figure 37. Example of BIOP

4. 系統架構設計與實作

在本論文當中，實作與實驗的系統，主要是參照 13818-1 的規格書、13818-6 的規格書、以及DVB-MHP的規格書中所制定相關的部份加以整理，並且考慮將子系統的各個部份整合時的效能考量，所設計出來的一套可以各種平台(包括windows及Unix like)上獨立執行的系統。根據實作時的作法以及效能的考量，整個系統架構如Figure 38所示。

其中主要分成三個部份：

1) TS Parser，處理 transport stream 的 packets 以及跟 Service Information 相關的部份，包含 Demux 和 Section Filter 中與 packets 處理相關的功能、以及 Service Information 中對於 PAT 及 PMT 處理的功能。

2) AIT Decoder，包含 AIT 的剖析，以及 application manager 中查詢 AIT 與 DVB-J Applicatoins 的功能，用來確認我們的設計是實際可行的。

3) DSM-CC，包含了 DSM-CC 以及 Section Filter 的部份，負責整個 object carousel 由接收到解碼以及儲存的處理，從 Sections 的處理直到完整的解出資料來爲止。

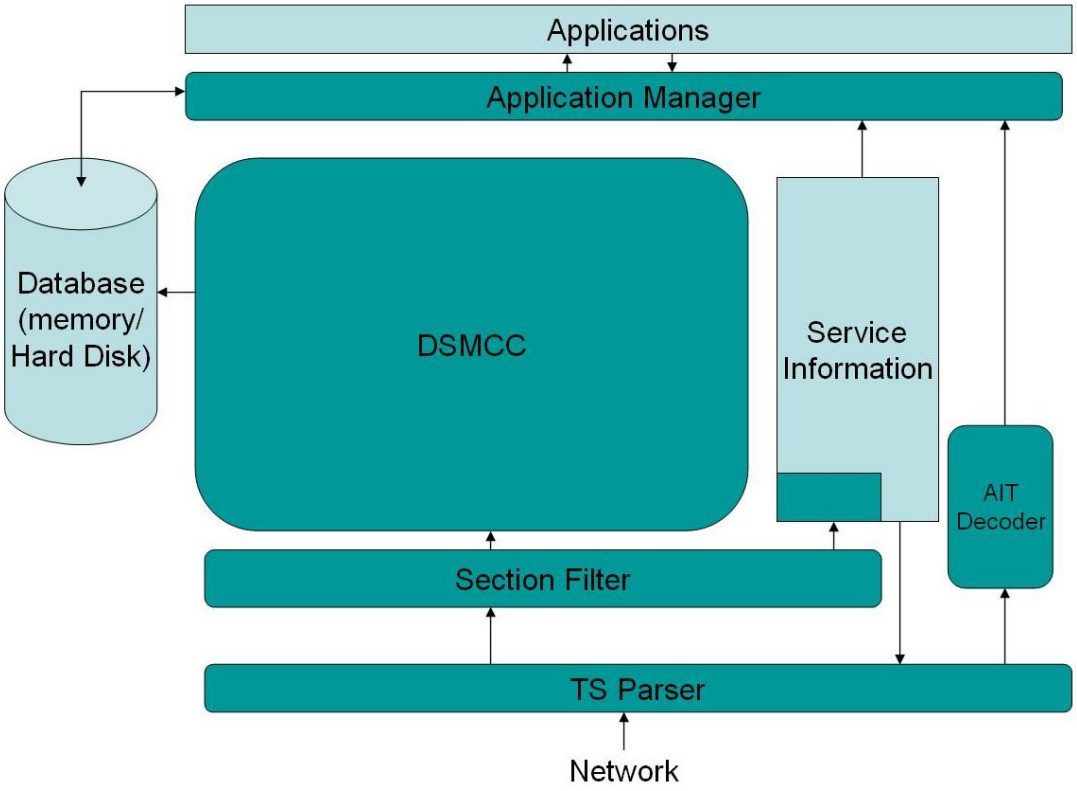

Figure 38. System Architecture of DSM-CC decoder

4.1. Transport Stream Parser

在 Service Information 的工作當中，只有 PAT 以及 PMT 的部份和 DSM-CC 相關。所以如果把 SI 這部份獨立出來實作的話，除了增加實作上的困難之外，在資料的傳輸上的時間耗費以及解出來的資料在送給 Demux 處理之前的記憶體使用，都是一種不必要的負擔。

所以在這個部份，本論文特別把Demux和PAT、PMT的處理這部份，以及Section Filter中處理packets的部份合併成爲TS Parser，專門處理將AIT的資料及DSM-CC的資料抽取出來的工作。流程大致上如Figure 39所示。

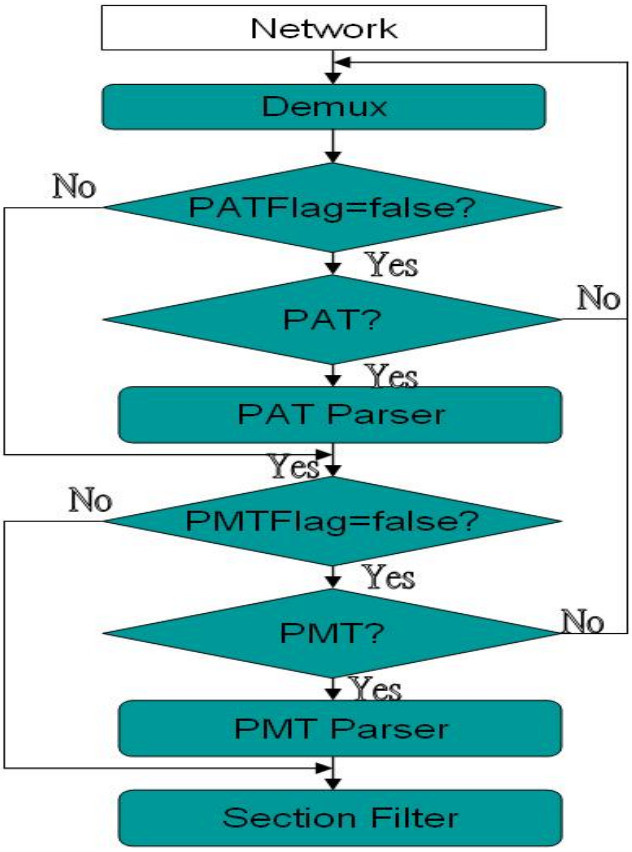

Figure 39. Block Diagram Transport Stream Parser

4.1.1. PAT and PMT

在本論文的實作系統當中，如果收到了一段 transport stream 的話，就會開始處理去讀出每一個 packet 的 PID。然後在處理到 PID= 0x0000 的 packet 之前(也就是 PAT)，所有收到的 packet 都全部忽略不繼續處理。

等到取得PAT的資料並且完成處理之後，我們可以透過PAT的內容得知PMT的PID，這時就開始處理PMT的內容，並且建立不同的table來儲存DSM-CC資料的PID以及AIT資料的PID。如Figure 40所示。

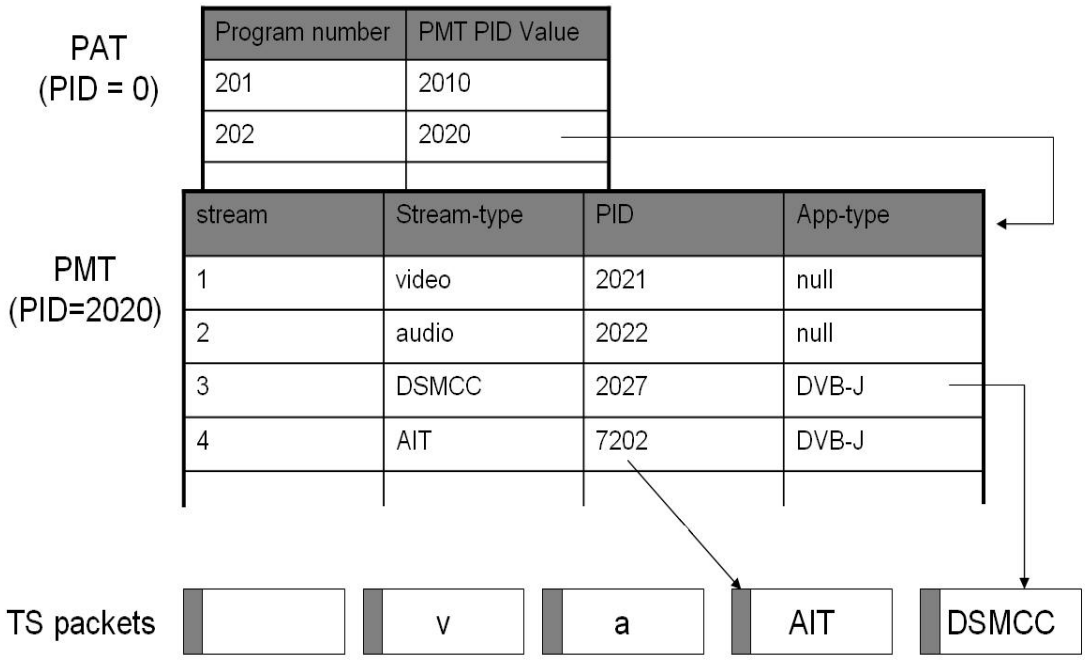

Figure 40. The procedure of PAT and PMT parsing

4.1.2. Section Filter

在取得 DSM-CC 跟 AIT 資料的 PID 之後，我們會開始讀取該 PID 對應的 packets。

以AIT爲例，首先先檢查這個packet的PID是不是屬於AIT，如果不是的話，就把整個packet丟掉，再往下讀一個packet。如果是屬於AIT的話，則檢查它的payloadUnitStartIndicator的值。如Figure 41所示。

1) 如果 payloadUnitStartIndicator = 0，表示在這個 packet 當中，沒有新的 section 的開頭，則檢查它相對應的 temp file 是不是空的。

   A. 如果 temp file 是空的，代表這個 section 還沒有被處理過，但是若是一個新的 AIT section 的話，那它的 payloadUnitStartIndicator 應該會被設爲 1，由此可知，這其中應該有丟失封包，所以我們必須把這個 packet 整個略過不處理，直接去讀取下一個 packet。

   B. 如果 temp file 不是空的，代表我們之前曾經處理過這個 AIT section

的值，則我們要作的就是把資料讀入 temp file，再去讀取下一個 packet。

2) 如果 payloadUnitStartIndicator = 1 的話，表示在這個 packet 中，存在一個新的 section 的開頭，這時我們就要去檢查 pointerField。

A. 如果 pointerField = 0 的話，代表上一個 section 的資料已經結束了，這個 packet 從頭開始都是一個新的 AIT section 的資料，所以我們就要把 temp file 關上，送給下一個處理者，並且再開一個新的 temp file，準備接受新的 section 的資料。

B. 如果 pointerField = n ≠ 0 的話，代表這個 packet 在 pointerField 後的第 n 個 byte 之後是屬於一個新的 section，則我們先寫入 n 個 bytes 的資料到 temp file，然後把 temp file 關上並送給下一個處理者，最後再開一個新的 temp file 把剩下的 bytes 寫入新的 temp file。

當我們成功的讀取出一個 section 之後，首先要檢查它的真實長度跟它的 section length 是不是一樣，因爲有些 sections 被切成 188 bytes 長的 packet 時，會因爲長度不夠，而在最後被加上若干個 padding bytes，所以我們要先把這些 padding bytes 過濾掉。

而 DSM-CC 也是透過類似的方法處理。最後再把這些完整的 sections 送到 Cyclic Redundancy Check 處理，經過確認這些 sections 的內容都正確之後，就可以分別送給 AIT Decoder 跟 DSM-CC 處理了。

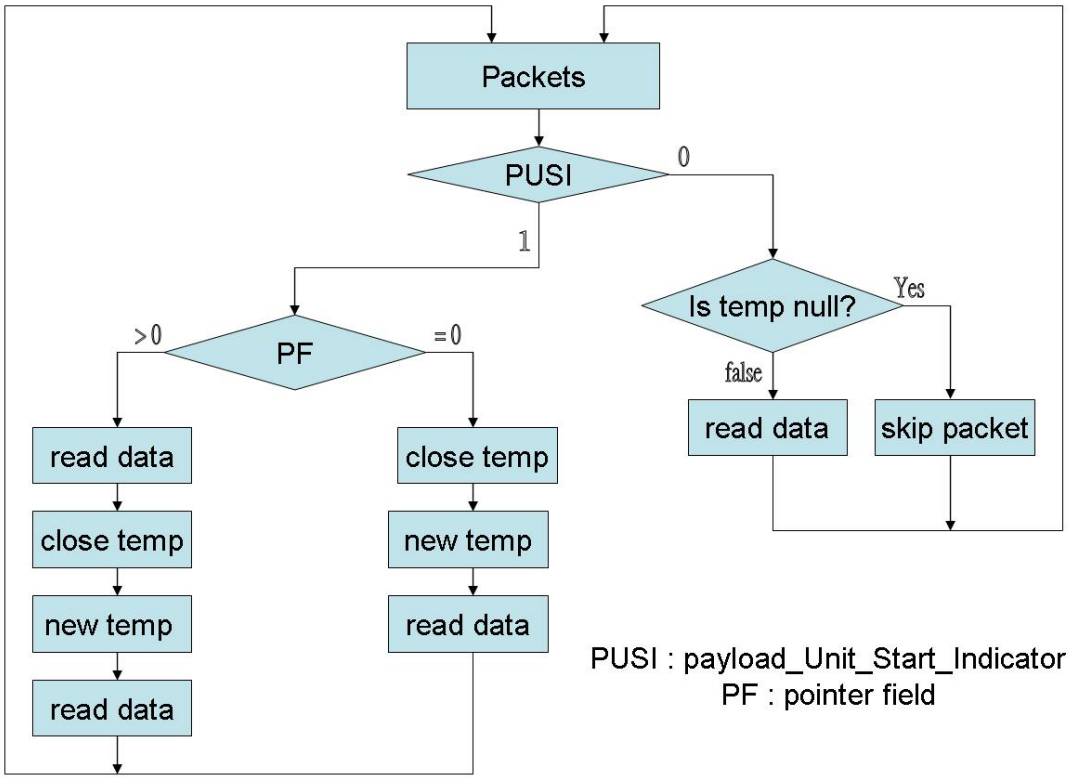

Figure 41. Scheme of TS parser

4.2. AIT Decoder

AIT Decoder 在本論文當中，主要分成下面兩個部份。

1) 處理 Application Information Table，將資料解碼，並設計一個 API 讓 Application Manager 可以主動讀取我們解出來的資料。

2) 用來驗證我們設計並實作出來的 AIT Decoder 是確實可以正確執行的簡易 Application Manager。

4.2.1. AIT Decoder and API

當我們處理過AIT的Header之後，就開始檢查是不是已經讀到table的結尾(EOF)了。如果已經到了table的結尾，那就跳到finished的state，並且中止AIT Decoder 這個程式。如果還沒到table的結尾，那就往下面讀取一個byte(descriptor_tag)，判斷它是屬於哪一種descriptor，並送給相對應的處理者解

碼，並且把解出來的資料存到各自的list structure當中，並跳回到EOF的state，並重覆此一步驟，直到我們讀取到table的結尾爲止。如Figure 42所示。

至於解出來的資料的儲存方式，則用一個static的LinkedList來記錄所有的資料，包含了name，application type，application location等資訊。而上面的每一個欄位都是一個String Array。使用static LinkedList的原因是爲了保證當這一個AIT Decoder結束時，資料不會隨之被清除。如此application manager才可以在需要時透過API取得資料。儲存格式如Figure 43所示。

提供給application manager的Interface則如Figure 44所示，當application manager需要取得目前所有能夠執行的應用程式的資料時，只要呼叫getAllAIT()這個function就可以了。

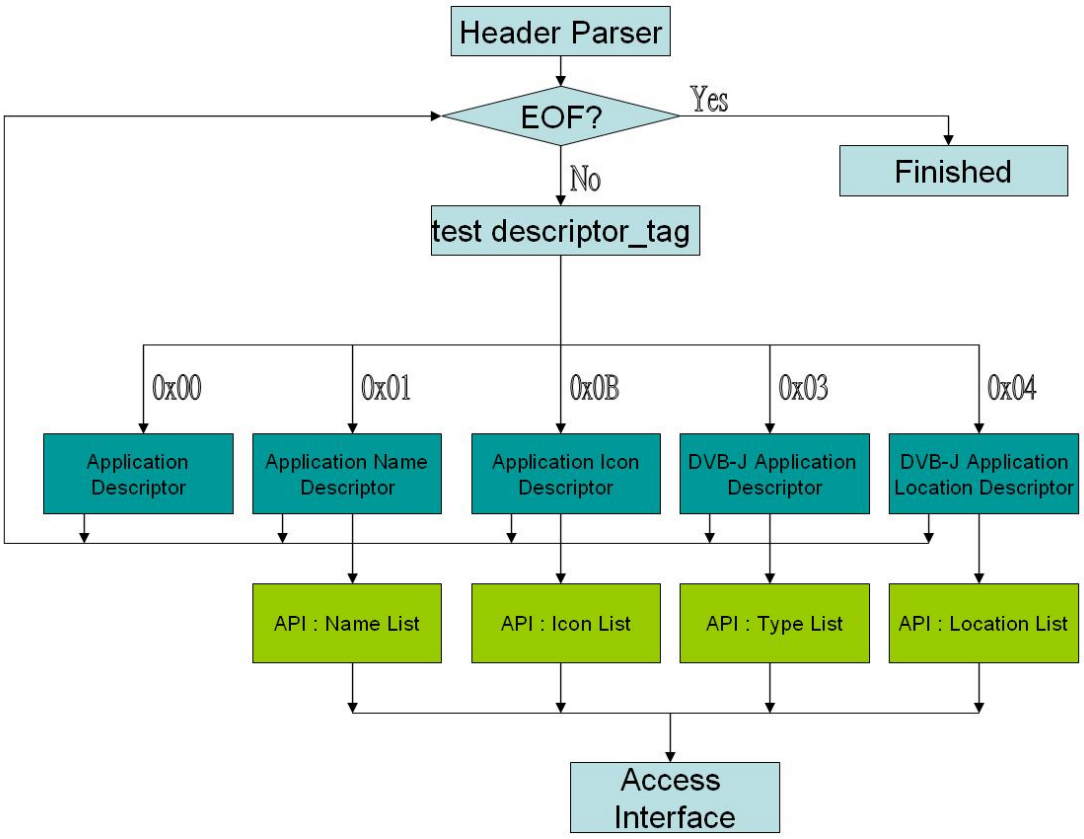

Figure 42. Block Diagram of AIT Decoder

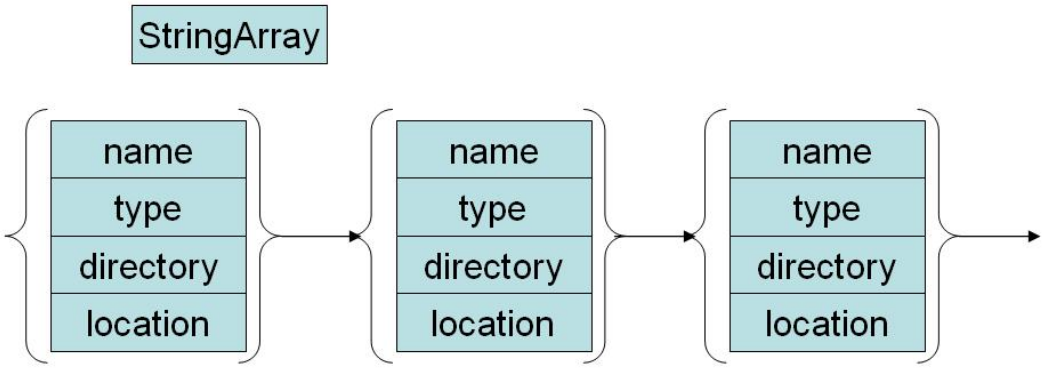

Figure 43. storage structure

```
public static AITDecoder[] getAllAIT()
{
        Object objArray[] = aitList.toArray();
        AITDecoder ret[] = new AITDecoder[objArray.length];
        for(int i = 0; i < ret.length; i++)
        {
            ret[i] = (AITDecoder)objArray[i];
        }
        return ret;
}
```

Figure 44. Interface of AIT Decoder

4.2.2. Application manager

因爲目前並沒有完整的 application manager 的 free ware 或 share ware, 如 OpenMHP 等平台所提供的 application manager 皆只是徒具形式的程式外殼。爲了測試本論文所提出的 AIT Decoder 可以正確執行，以及 API 的正確性，我們另外實作了部份的 application manager 功能，用來呈現 AIT 的內容以驗證 AIT Decoder 的正確性。

作法是先宣告一個ait的陣列，然後呼叫getAllAIT()這個函式，把我們已經解出來的資料全部從AITDecoder端取得，然後再一個個的存到list當中，最後畫在

畫面上。如Figure 45所示。

```
AITDecoder ait[] = AITDecoder.getAllAIT();
for(int i = 0 ; i < ait.length ; i++)
{
        for(int j = 0; j < ait[i].name.length; j++)
        {
                list1.add(ait[i].name[j]);
        }
}
```

Figure 45. Function in Application Manager

4.3. DSM-CC

Transport Stream 的 packets 經過 TS Parser 的處理之後，會產生一個個的 sections，然後送給 Section Filter 處理，轉成 blocks 之後，透過 table_id 及 table_id_extension 這兩個欄位來判斷它們是屬於 DSI、DII、DDB 中哪一種類型的資料並送給相對應的 Decoder 處理。

DDB blocks的資料，會根據DII block當中所記錄的資訊重新組合成爲modules，最後再透過BIOP Decoder轉回原本的資料並儲存到記憶體當中。DSM-CC的執行流程大致上如Figure 46所示。

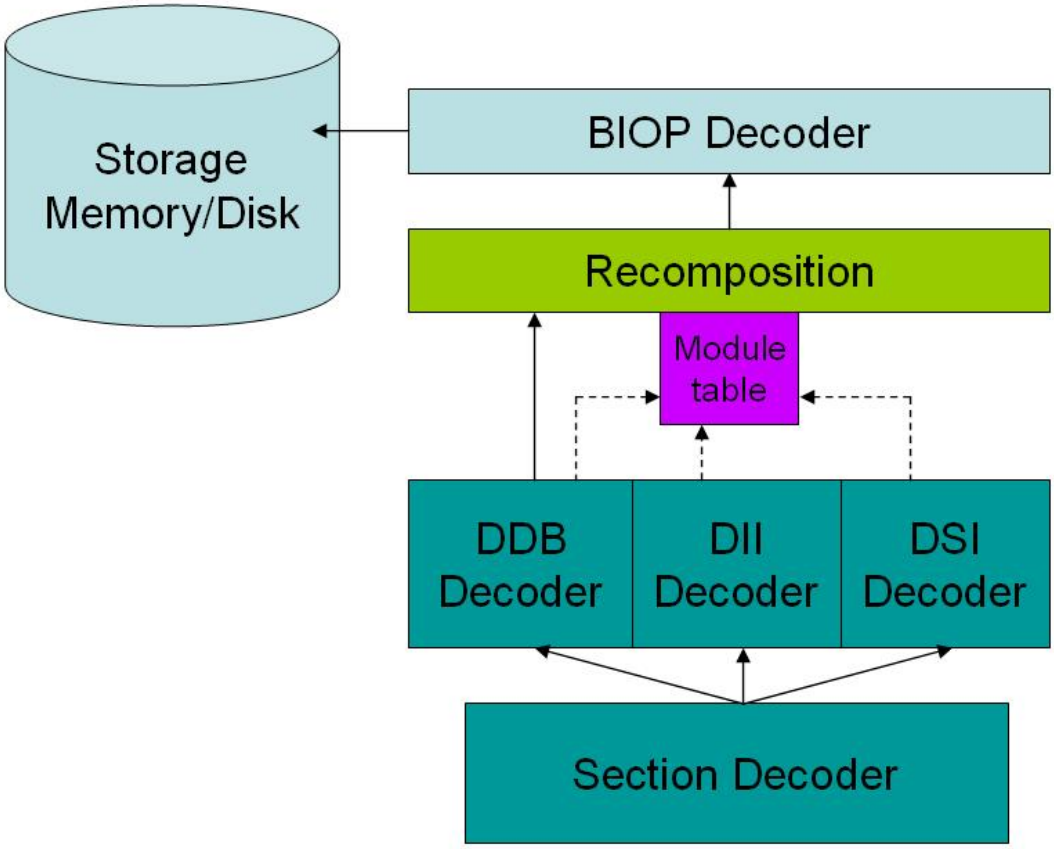

Figure 46. Block diagram of DSM-CC

因爲所有跟 modules 相關的資訊都記錄在 DII 當中，所以要等到我們處理過 DII section 之後，才能有效率的判斷這一個 DDB 是否已經有效率的處理過。又因爲每一個 service 都只會有一個相對應的 DII section，所以我們只需要處理一次就可以了。對此，我們特別設置了 diiFlag 以及 ddbFlag 兩個旗幟。

1) diiFlag：因爲每一個 service 都只會有一個 DII，所以利用 diiFlag 來判斷是不是已經處理過 DII section 了，如果 diiFlag = true，則我們可以直接把之後收到的 DII section 略過，以節省處理時間。

2) ddbFlag：若 ddbFlag = false 的話，代表我們還沒有收到 DII，這時就算解出了 DDB，也很難去記錄我們已經處理過哪些 modules 的哪些 blocks。所以在處理 DII 之前，我們把 ddbFlag 設爲 false，並且略過在此時收到的所有 DDB sections。

在處理過DII section之後，我們可以取得block size，number of modules，module ID，跟module size這幾個資訊。利用$^{\text{moduke size}}/_{\text{block size}}$，我們可以知道每一個module當中有幾個blocks，並依此建立起moduleTable跟blockCount這兩個table。

1) moduleTable：第一個欄位是moduleId，後面的欄位數則跟block number一樣。當我們收到一個DDB之後，解出它的moduleId跟blockNumber之後，就可以到moduleTable作比對判斷這個DDB是不是已經處理過了。如果還沒處理，才儲存起來，並且把它相對應的欄位設爲true，否則就直接略過。moduleTable如Figure 47所示。

2) blockCount：第一個欄位也是 moduleId，第二個欄位則是一個 counter，記錄了這一個 module 所包含的 blocks 的數量，當我們每收到這個 module 的一個 DDB，比對過它是還沒處理過的，就處理它，然後把這個 module 相對的 counter - 1，直到 counter = 0 時，我們就知道這個 module 所有的 block 都已經收到了，就可以送給下一個 Decoder 處理。

$moduleId_1$	$block_1$	$block_2$	$block_3$	$block_4$	
$moduleId_2$	$block_1$	$block_2$	$block_3$		
$moduleId_3$	$block_1$	$block_2$	$block_3$	$block_4$	$block_5$
$moduleId_4$	$block_1$	$block_2$	$block_3$	$block_4$	$block_5$

Figure 47. Example of Module Table

每當我們收到一個block，就把它的moduleId、blockNumber、data這三項存到我們自己的資料結構DataRecord當中，如Figure 48所示，並且把blockCount中相對應的counter – 1。

當counter = 0 時，我們知道這個欄位相對應的module已經全部收到了，這時

我們會再呼叫另外一個我們自己的資料結構MDSComparator，如Figure 49所示，它會對相同moduleId的所有bolck的blockNumber作排序，然後我們再依序從陣列當中，把屬於我們需要的module的每一筆資料抓出來儲存，就可以得到一個完整的module了。

```
class DataRecord
{
	protected int moduleId;
	protected int blockNumber;
	protected byte data[];

	protected DataRecord(int moduleId, int blockNumber , byte data[])
	{
		this.moduleId = moduleId;
		this.blockNumber = blockNumber;
		this.data = data;
	}
}
```

Figure 48.　My Data Structure：DataRecord

```
class MDSComparator implements Comparator
{
	public int compare(Object o1, Object o2)
	{
		return ((DataRecord)o1).blockNumber-((DataRecord)o2).blockNumber;
	}
}
```

Figure 49.　My Data Structure：MDSComparator

每當我們收到一個完整的 module 之後，就可以把它送給 BIOP Decoder 處理，而 BIOP Decoder 再根據收到的是哪一種類型的資料，分別呼叫不同的 decoder 作處理。

如果收到的是Srg的話，那就把旗標isSrg設爲true，然後呼叫BIOP DirMessage Decoder，如果是dir的話，那就把旗標isSrg設爲false，然後呼叫BIOP DirMessage Decoder。而 BIOP DirMessage Decoder 會處理整個資料的內容，並且把

objectKeyDataByte的內容，還有這個資料結構中所包含的所有資料的name、type、profileObjectKeyDataByte存進我們自己的FileNode資料結構當中。FileNode的結構如Figure 50所示。

如果收到的是 Fil 的話，先把目前已經收到並記錄的 FileNode List(fnList)轉成陣列(treeArray)，再分成兩個步驟。第一步是重新建立目前已經收到的節點的樹狀結構，第二步是將找到的資料，存到它相對應的檔案當中。

1) 先從 FileNode 中找出 isSrg = true 的點，把它設為 root，丟進 queue 中，再遞迴的檢查它每一個子節點是否已經在 fnList 當中。如果否的話，表示這個子節點我們還沒處理過，就準備處理它。

   - 如果子節點的資料類型是檔案的話，那我們就把它的 profileObjectKeyDataBye、Type、numberOfChildren 存起來，然後加到 fnList 當中。
   - 如果子節點的資料類型是資料夾的話，那我們就拿子節點的 profileObjectKeyDataBye 去和 treeArray 中每一個節點的 objectKeyDataBye 比較。如果相同的話，表示 treeArray 中的這一項就是這個子節點。

2) 如果這個子結點的類型是資料夾的話，那我們就找出由根節點到這個子結點的路徑，然後把資料夾建立。如果子結點的類型是檔案的話，就先到去檢查它的資料是不是已經在 missingList 當中，如果是的話，就建立檔案然後把它的資料從 missingLsit 中搬過來。

以上步驟重覆到 queue 中已經沒有資料為止。

此外，BIOP FileMessage Decoder 在處理一筆資料時，會先去檢查目前 fnList 中所有的檔案結點中，是否有結點的 profileObjectKeyDataBye 和這份資料的 objectKeyDataBye 相同。如果有的話，就把資料存進這個檔案當中；如果沒有的

話，那就用這個結點的 objectKeyDataBye 作爲檔名建立 tempFile，然後把檔案丟進 missingList 當中。

BIOP的執行流程如Figure 51所示。

```
class FileNode
{
	protected boolean isSRG;
	protected int objectKey;
	protected int type;
	protected String name;
	protected int bindingsCount;
	protected String childName[];
	protected int childType[];
	protected int childKey[];
	protected FileNode child[];
	protected FileNode parent;
	protected String pathname;
	protected File file;

	protected FileNode(int objectKey , int type, int bindingsCount)
	{
		isSRG = false;
		this.objectKey = objectKey;
		this.bindingsCount = bindingsCount;
		childName = new String[bindingsCount];
		childType = new int[bindingsCount];
		childKey = new int[bindingsCount];
		child = new FileNode[bindingsCount];
		parent = null;
		pathname = null;
		file = null;
	}
}
```

Figure 50.　My DataStructure：FileNode

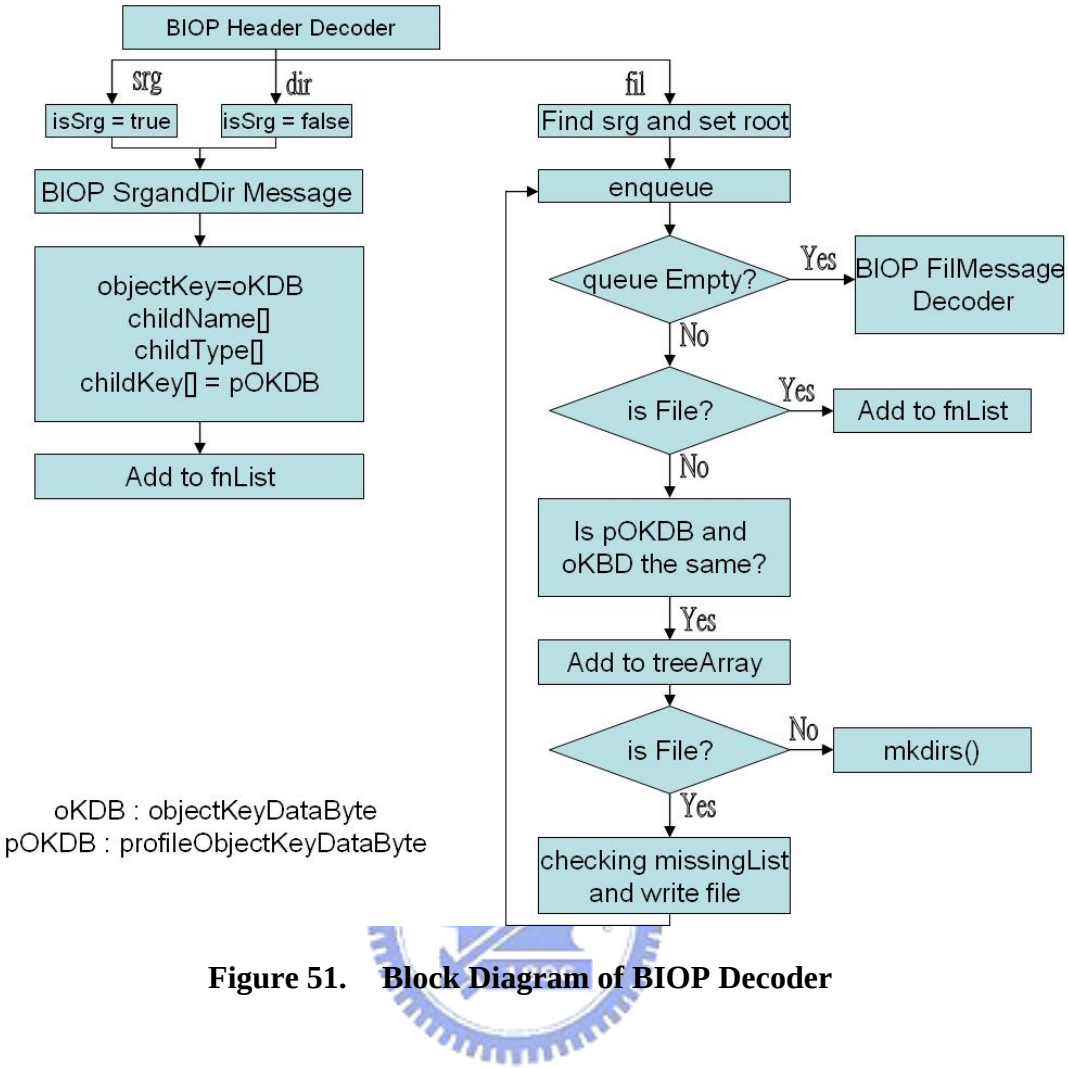

Figure 51. Block Diagram of BIOP Decoder

5. 實驗結果

在本章當中，我們進行一些實驗來證明我們的系統是可以正確執行的。

5.1. Environment of Experiments

一、硬體平台。

我們使用的硬體平台是Xilinx ML310 硬體開發板[14]，上面有PowerPC 405跟FPGA。PPC的時脈可以達到 300 MHZ，在平台上執行的作業系統是MontaVista Linux 3.1 Pro。

二、Java Virtual Machine

我們所使用的JVM則在是Sun公司所release的Connected Device Configuration (CDC)中的CVM[15]。其中CVM主要是Sun公司針對消費性產品及嵌入式平台所開發出來，符合Java™ 2 Platform, Micro Edition (J2ME™ technology)標準的虛擬機器。

5.2. Design of Experimentation

實驗的步驟大致上分爲下面三個：

1) 選擇適當的資料，並且將它們包裝產生 Trasnport Stream。
2) 把我們所產生的 Transport Stream 餵給 DSM-CC Decoder 處理，重新解回原始的資料型態。
3) 把我們解出來的資料，拿到IRT的MHP reference implementation執行[16]，驗證解出來資料的正確性。

5.2.1. Transport Stream Generation

我們用來產生Transport Stream的軟體是由CINECA所開發提供的JustDVB-IT

1.1.1[17]。產生TS的過程大致如下：

1) 產生 Audio/Video 的 Packetized Elementary Stream (PES)，並且轉成 TS packets。

- 利用 MainConcept mpeg2 encoder[18]這套軟體，分別產生 Audio 及 Video 的 elementary stream。
- 利用 JustDVB-IT 中，esaudio2pes 及 esvideo2pes 這兩個軟體，把 elementary stream 轉成 PES
- 利用 JustDVB-IT 中，pes2ts 這個軟體，把 PES 轉成 TS packets。

2) 產生包含有資料的 DSM-CC sections，並且轉成 TS packets。

- 先把我們要轉成 sections 的資料，建立它的樹狀結構，或者用它原本的結構。
- 利用 JustDVB-IT 中，oc2sec 這套軟體(其中包含 file2mod 跟 mod2sec 兩個程式)，從檔案轉成 sections。
- 利用 JustDVB-IT 中，sec2ts 這套軟體，把 sections 轉成 TS packets。

3) 產生 Service Information 必要的資訊。

- 在 JustDVB-IT 這套軟體當中，它主要的程式是以 Python 這個語言撰寫的，所以要產生跟 SI 相關的資訊，要從它原本附的範例程式來加以修改，使之配合我們的測試資料。
- 我們主要需要的資訊是 PAT，PMT，跟 AIT 這些。

4) 把這些資訊結合在一起產生 Transport Stream。

- 原本的作法是把以上的資訊，通通透過 JustDVB-IT 中，tscbrmuxer 這個軟體，mux 成一個完整的 TS，然後透過 Dektec

生產的 DTA-102 這張卡把 TS 傳送出去。不過因爲我們主要是爲了測試產生的 TS 以及我們的解碼過程是正確的，所以就直接把產生的訊號存檔然後送給我們的 Decoder 處理。

5.2.2. Correctness of DSM-CC Decoder

爲了驗證我們實作的 DSM-CC Decoder 的正確性，實驗步驟分爲下面兩種：

1) 對於產生的 TS，重新解回原本的檔案格式，並且驗證樹狀結構被完整重建。

2) 將解出來的應用程式拿到 IRT 的 reference implementation 執行，驗證執行的過程及結果正確。

5.3. Result of Experimentation

在這個實驗當中，我們設計了三個不同的實驗內容。

5.3.1. Case 1

實驗用的資料是在 IRT 的 reference implementation 當中，可以執行的一個小遊戲：貪食蛇；實驗用的 audio/video 是隨意取得的一段影片檔，分別將音訊及影像擷取出來。用來測試我們可以順利的重建原本的資料結構，實驗用的 Transport Stream 是利用 JustDVB-IT 產生的。

1) 產生 audio/video 部份的 transport stream。

- 先利用在 windows 上執行的 Mainconcept MPEG Encoder 分別產生 MPEG-2 的 Audio Elementary Stream(ES) 及 Video Elementary Stream。
- 利用JustDVB-IT的工具把ES轉成PES，指令如Figure 52所示。
- 再利用工具由PES轉成TS，指令如Figure 53所示。

2) 產生 data 的 transport stream。

- 首先由IRT的資料中，找出一個可以執行的xlet：snake，資料結構如Figure 54所示。
- 接下來再利用軟體將之轉成sections，指令如Figure 55所示。
- 再由sections轉成transport stream，指令如Figure 56所示。

3) 編寫 SI 相關的資料內容，PAT、PMT、AIT。

4) 將上面的資料mux成完整的transport stream，如Figure 57所示。並送給我們的程式執行。

5) 驗證確實可以重新建立多層的樹狀結構。執行結果如Figure 58、Figure 59所示。

```
Usage: 'esaudio2pes audio.es stream_id pts_step audio_frame_size'
./ esaudio2pes audio.es 192 2160 768 > audio.pes
Usage: 'esvideo2pes video.es stream_id pts_step'
./ esvideo2pes video.es 224 3600 > video.pes
```

Figure 52. Step of ES to PES

```
Usage: 'pes2ts file.pes pid'
./pes2ts audio.pes(video.pes) 2006 > audio.ts(video.ts)
```

Figure 53. Step of PES to TS

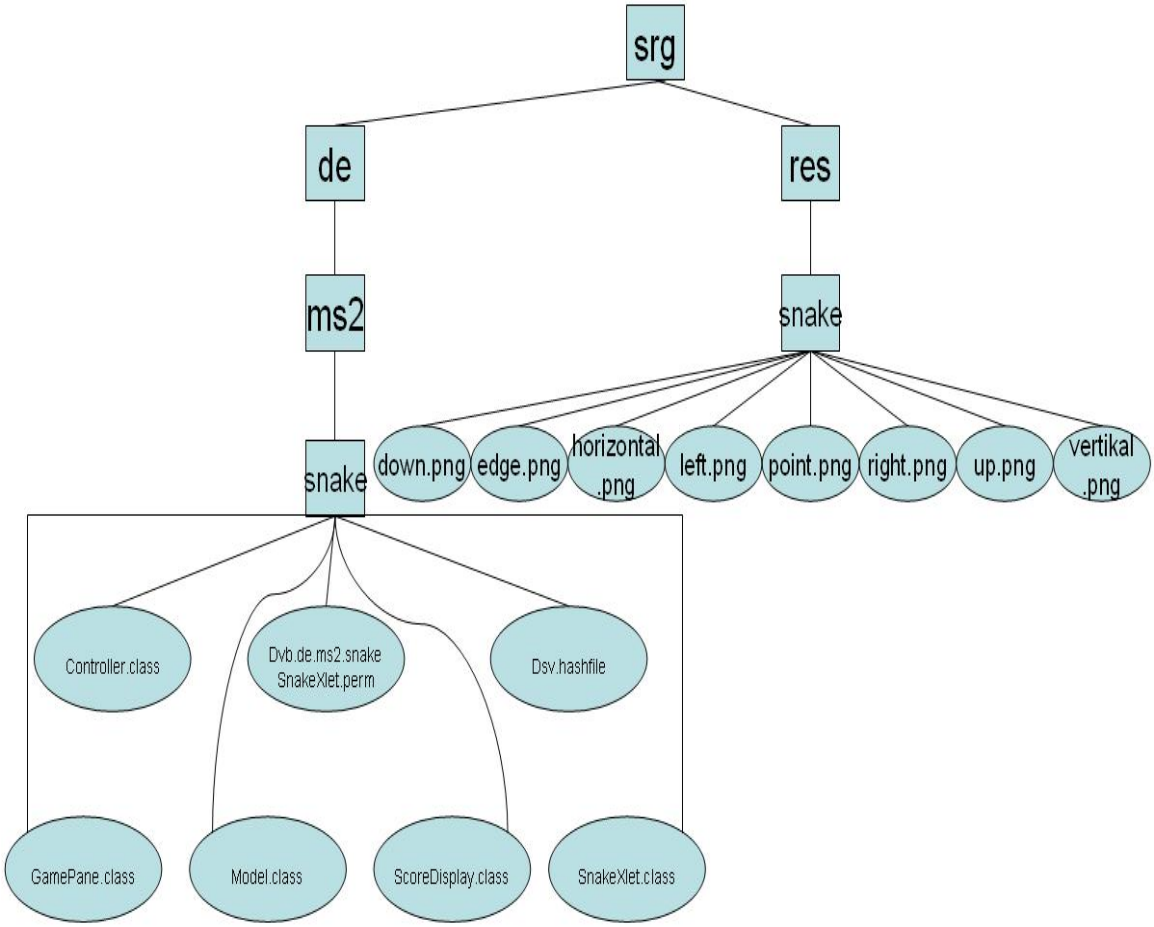

Figure 54. Experimentation 1

```
Usage: ./file2mod.py <InputDirectory> <OutputModulesDirectory>
download_id carousel_id association_tag version
./file2mod.py srg/ test/ 1 2 0xC 1
Usage: ./mod2sec.py <InputModuleDirectory> <OutputSectionsDirectory>
./mod2sec.py test/ case/
```

Figure 55. Step of files to sections

```
Usage : ./sec2ts PID < OutputSections > OutputTS
./sec2ts 2000 < oc.sec > oc.ts
```

Figure 56. Step of sections to TS

```
Usage: 'tscbrmuxer b:bitrate_pat pat.ts b:bitrate_pmt pmt.ts b:bitrate_ait ait.ts b:bitrate_data data.ts ...
./ tscbrmuxer b:100 pat.ts b:100 pmt.ts b:100 ait.ts b:100 oc.ts b:44800 audio.ts b:44800 video.ts > TransportStream.ts
```

Figure 57. Step of TS mux

```
[frozenn@crow tree-1.5.0]$ ./tree ../../srg/
../../srg/
|-- de
|   `-- ms2
|       `-- snake
|           |-- Controller.class
|           |-- GamePane.class
|           |-- Model.class
|           |-- ScoreDisplay.class
|           |-- SnakeXlet.class
|           |-- dvb.de.ms2.snake.SnakeXlet.perm
|           `-- dvb.hashfile
`-- res
    `-- snake
        |-- down.png
        |-- edge.png
        |-- horizontal.png
        |-- left.png
        |-- point.png
        |-- right.png
        |-- up.png
        `-- vertikal.png
5 directories, 15 files
```

Figure 59. Screen Shot of Experimentation 1

5.3.2. Case 2

我們隨意產生了一筆資料，其中包含有，影片、mp3，圖片，java class file。並不是一個可以執行的應用程式，目的是爲了測試當 TS 包含了各種不同類型的資料，然後產生多層的數狀結構，我們的程式仍然可以正確的對它解碼。

資料的架構圖如Figure 60所示。執行之的檔案示意圖如Figure 61所示。

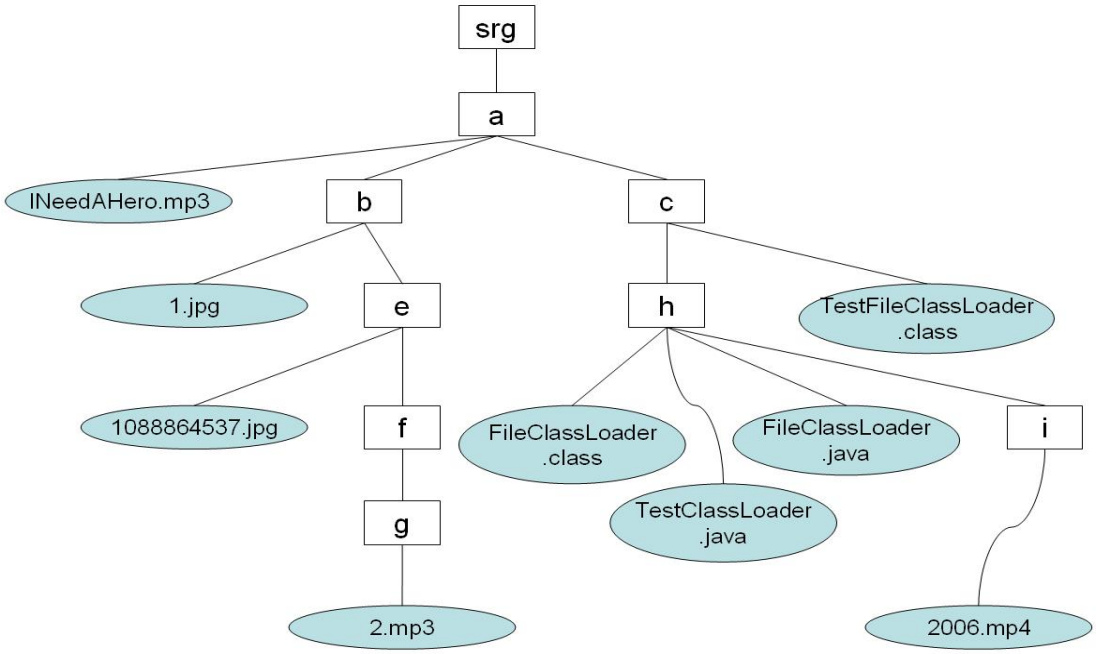

Figure 60. Experimentation 2

```
../a/
|-- INeedAHero.mp3
|-- b
|     |-- 1.jpg
|     `-- e
|           |-- 1088864537.jpg
|           `-- f
|                 `-- g
|                       `-- 2.mp3
`-- c
      |-- TestFileClassLoader.class
      `-- h
            |-- FileClassLoader.class
            |-- FileClassLoader.java
            |-- TestClassLoader.java
            `-- i
                  `-- 2006.mp4

7 directories, 9 files
```

Figure 61. Result of Experimentation 2

5.3.3. Case 3

實際可讀可播放的應用程式，用來驗證對於大檔案我們仍然可以正確的解讀，實驗用的資料是台北科技大學的郭天穎教授的實驗團隊從公共電視的數位訊號錄製的。

執行之後會產生四個應用程式，其中包括了Pizza Hut的點餐系統，之後我們再將這個應用程式拿到IRT的reference implementation執行，驗證解出來的資料正確無誤，執行的畫面如Figure 62，Figure 63所示。

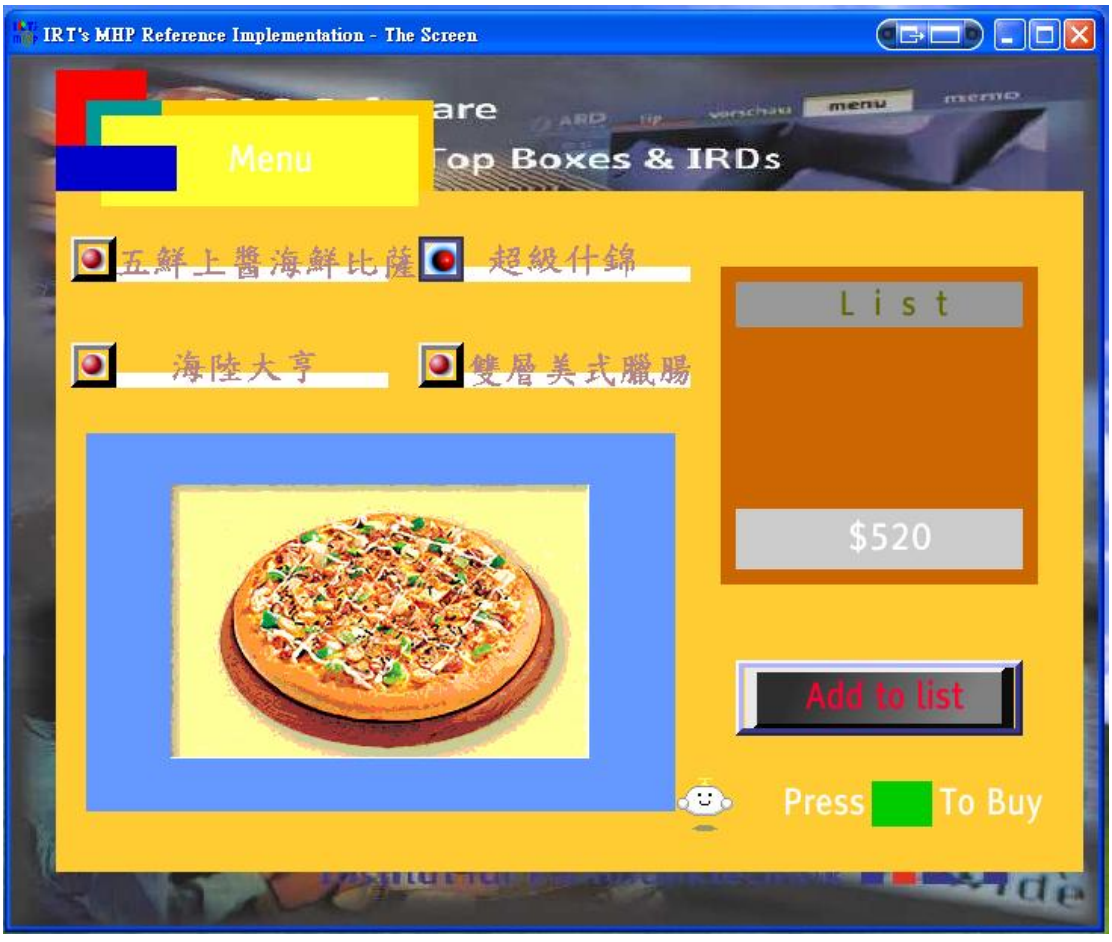

Figure 62. Screen Shot of Experimentation 2

Figure 63. Screen Shot of Experimentation 3

6. 結論與展望

6.1. 研究與實驗結果討論

本論文的主要目的是建立一個精簡且快速的 DSM-CC Decoder，從 Transport Stream的接收與處理，一直到解出完整而正確的資料提供給 Application Manager 讀取爲止。

在第五章的實驗當中，所有有問題的資料，都會在 sections 被重新組成之後，便由 CRC Check 給過濾掉，所以可以不用考慮在傳送時資料出錯的問題。我們除了測試並驗證多層的樹狀結構所產生出來的資料是可以被正確處理的之外，也透過 IRT 的實作，驗證了我們解出來的資料是正確的。

雖然本論文的實作系統因爲目前全部都是在軟體環境上執行的，所以在效能上可能比不上用硬體實作的作法，不過因爲本論文的實作系統符合Java™ 2 Platform, Micro Edition(J2ME)，所以可以很容易的在不同的嵌入式平台中移植，再配合上硬體的Java加速器，在整個效能上仍然有很大的進步空間。另外可以以現有的架構爲基礎，發展出可以處理不同型態的應用程式，如DVB-HTML，的DSM-CC Decoder，以期在更多不同的平台上執行。

6.2. 未來工作及展望

未來仍然有幾個方向要繼續努力的。

- 增進處理的效能。

  雖然在本論文的實作系統當中，並沒有用到很複雜的演算法，不過如果能夠有效的降低 IO 的次數，同樣也能夠在效能上得到顯著的改善。雖然在實作之初已經有考慮到效能的問題，不過當整個系統整合之後，效能上應該還有可以改良的部份。

- 建立 Version Number 的監視與控制。

  因爲目前我們所有的測試資料來源，都是自己產生或是從網路訊號上側錄而來的，所以不會有資料更新造成 Version Number 不同的問題，但是這種情況在真實的環境中是無可避免的，所以建立對 Version Number 的監視，並且適時的把舊的資料清除換上新的資料是當務之急

- 開發各種介面。

  爲了達到每個元件之間快速的溝通以及資料傳遞，可以再加以開發不同的 interface 來達到如上的目標，例如：AITDecoder 和 Application Manager 之間的溝通介面。

- 和 OpenMHP 的整合

  雖然本論文的實作系統可以完全獨立執行，不過要和一個 open source 的 MHP middleware 配合執行，才是一個完整的 Set Top Box 模擬系統，而 OpenMHP 是本論文實作系統最早的參考 middleware，所以目前考慮將系統和 OpenMHP 互相整合。

7. 參考文獻

[1] ETSI TS 102 812, “Digital Video Broadcasting(DVB); Multimedia Home Platform(MHP) Specification 1.1.1,” June 2003.

[2] ISO/IEC JTC1/SC29/WG11, “ISO/IEC 13818-1: Information technology – Generic coding of moving pictures and associated audio – System,” November 1994.

[3] ISO/IEC JTC1/SC29/WG11, “ISO/IEC 13818-6: Information technology – Generic coding of moving pictures and associated audio information– Extension for Digital Storage Media Command and Control,” November 1995.

[4] OpenMHP Home Page, http://www.openmhp.org/.

[5] Hongguang Zhang, Tianpu Jiang, Zhiqi Gu, Shibao Zheng, “Design and Implementation of Broadcast File System Based on DSM-CC Data Carousel Protocol,” *IEEE Transactions on Consumer Electronics*, August 2004.

[6] Eun-Jung Kwon, Han-Seung Koo, O-Hyung Kwon, Soo In Lee, “Object Acquiring Time Saving Scheme in Data Broadcasting,” *Consumer Electronics, 2005. ICCE. 2005 Digest of Technical Papers. International Conference on*, January 2005.

[7] Dong-Hwan Park, Tai-Yeon Ku, Kyeong-Deok Moon, “Real-Time Carousel Caching and Monitoring in Data Broadcasting,” *IEEE Transactions on Consumer Electronics*, February 2006.

[8] L. Atzori, M. Di Gregorio, “Multimedia Information Broadcasting Using Digital TV Channels,” *Broadcasting, IEEE Transactions on*, September 1997.

[9] C. Peng, P. Vuorimaa, “Digital Television Application Manager,” *Multimedia and Expo, 2001. ICME 2001. IEEE International Conference on*, August 2001

[10] Seven Morri, “The Interactive TV Web – Signalling MHP applications,” http://www.interactivetvweb.org/tutorial/mhp/app-signalling.shtml.

[11] Advanced Television Systems Committee, “ATSC Standard: Transport Stream File

System Standard,” February 2003.

[12] Digital Video Broadcasting, “ Digital Video Broadcasting(DVB); Implementation guidelines for Data Broadcasting,” June 1999.

[13] Seven Morri, “The Interactive TV Web – Object Carousels,” http://www.interactivetvweb.org/tutorial/dtv-intro/dsm-cc/objectcarousel.shtml

[14] Xilinx inc. http://www.xilinx.com/products/boards/ml310/current/

[15] Sun microsystem, http://www.sun.com/software/communitysource/j2me/cdc/

[16] IRT GmbH, Munich , http://www.irt.de/IRT/mhp/mhp-e.htm

[17] Cineca DVB software, http://www.cineca.tv/labs/mhplab/index.html

[18] Main Comcept Home Page, http://www.mainconcept.com/site/