第二章

相關研究

2.1 多人虛擬環境系統

[44]中曾對虛擬環境這個名詞加以定義。這篇論文指出，虛擬環境指的是一個互動的、擬真的、使用者界面多元化的人造環境。目前的虛擬環境主要是基於下列三大技術的發展: [34]

1. **輸入與偵側(sensor)技術:**在和使用者互動過程中，取得輸入資訊，並傳送給系統，例如手勢偵測(Gesture recognition)、語音辨識(Speech recognition)等等。

2. **輸出與顯示技術:**將系統狀態與資訊輸出以各種方式輸出給使用者，例如 3D 頭罩、螢幕顯示與語音合成(Speech synthesis)等。

3. **計算與支援技術:**泛指用來支援虛擬環境運算的技術，例如知識表達(Knowledge representation)、對話管理(Dialog management)、虛擬環境中人物與物件之行為管理等。

多人虛擬環境系統(Multi-user Virtual Environment,MUVE)則是結合了虛擬實境的技術與分散式系統的特色，讓網路上的使用者利用各自的虛擬人物，在一個擬真的虛擬環境中進行互動。以往由於計算機計算能力的限制與網路頻寬的不足，實作成本非常高，所以無法普及[49]。近年來在這二個領域有很大的進展，因此陸續有許多多人虛擬環境系統被應用到軍事與教學用途上[5] [50]。

多人虛擬環境中最大的特色是使用者不只能和虛擬環境中的物件互動，還要能彼此互動。每個使用者對虛擬環境的改變必須是可觀察的(observable)，在同一個世界中每個使用者所看到的情況也必須一致。

依據訊息傳播的方式，我們主要可將多人虛擬環境系統分為二大類，即*主從式架構(Client-Server)*與*點對點(Peer-to-peer)架構*二種。例如[1]與[7]採用了主從式架構設計，其主要做法是每個 Client 將自己的相關訊息藉由 Server 廣播至其它 Client。主從式架構的好處是容易實作，但當 Client 端數量增多時，很容易造成效能的瓶頸。而[19]與[22]則是採用點對點架構，他們則針對主從式架構效能上的問題，將原本由 Server 負責的訊息傳送協調功能分散給每個 Client 端來做，但這種方式最大的問題在於每個 Client 實作的機制較為複雜。

這二種架構各有其適用的場合。我們認為由於近年來網路與硬體的進步相當快速，在 Client 數目不多的情況下，多人虛擬環境的效能問題得以緩解，再加上主從式架構具有容易實作且擴充的特性，所以本研究中所使用的是採用主從式架構的 IMNet(Intelligent Media Net)[26]。IMNet Server 與 IMNet Client 與 VNet[46]設計理念類似，是由政大資科系智慧型媒體實驗室研發的多人虛擬環境系統。

2.2 腳本動畫語言

目前大部份動畫的產生是以動畫設計軟體(如 3D MAX 及 MAYA 等)或運動抓取(Motion Capture)的方式完成。最後並以動畫格中個別物體座標轉換的低階方式儲存動畫結果，並以這些應用軟體獨有的原始格式存在。

另一種表示動畫的方式是使用動畫腳本(Animation Script)。早期動畫設計軟體還沒風行時，就有許多相關的研究出現。互動式的動畫腳本則是從 Improv [39]開始就許多的相關研究投入這項議題。針對人體的動畫描述還有 Avatar Markup Language(AML) [30]和 STEP [25] 。

AML 主要的動作是由一個 bap 檔案做描述，使用者只能改變速度、強度等有限屬性，對整體的動作更改的能力很薄弱。AML 臉部動畫和語音的表現相當重視，例如臉部表情可以用「情緒」這種抽象的字眼來做出不同的效果；而 STEP 的特色在於能將自動推理的功能與電腦動畫的結果進行整合。STEP 是一個專注在肢體的動畫語言，可以藉由組合定義出新動作。不過 STEP 的動畫的豐富性並不強，動畫指定完成就無法去更改其強度、動畫撥放時間等動畫屬性。

XAML(eXtensible Animation Markup Language)[33]也是一個專注在肢體的腳本動畫語言，這套語言讓使用者可以彈性地使用不同層次的方式指定虛擬演員的動作。藉由 XAML，使用者可以重複利用之前創造的動畫元件來產生出一連串的動畫，或者是可以更改已經定義好的動畫的部分內容來產生一套新的動畫。XAML 語法採用 XML 格式，是一套可擴充性相當高的腳本動畫語言。

2.3 語音合成與語音辨識技術

語音技術主要分成輸出及輸入二部份，使用語音技術的應用程式往往需求很大的系統資源。在 1990 年代初期，電腦硬體技術開始突飛猛進後，原本只能在大型主機上實驗的語音辨識引擎便開始被使用在個人及商用電腦上。

2.3.1 語音合成(Speech Synthesis)

在做語音輸出時，可尋求專業錄音人員預先錄製好的音檔。但輸出內容一旦改變，就要重錄，十分麻煩。所以就有了可動態即時合成語音的語音合成器(Speech Synthesis,或稱 Text to Speech, TTS)的發展。TTS 可以讀取輸入的文字，依據設定的規則轉換成語音音檔，如此一來便能輕鬆改變音檔內容，但目前大部份 TTS 所合成的語音聽起來畢竟還是不像人類直接說話來得自然。所以在目前的語音應用程式中，預錄音檔多負責較「不易改變」的語音輸出，而變動性很大的字句(如姓名、數量等)，則通常交由 TTS 來產生。

文字合成語音的過程主要可分為下列五個步驟[28] :

1. **語法結構分析 (Structure analysis):** 處理輸入的文字，決定該段文字的段落、句子的起點與終點。
2. **文字前處理 (Text pre-processing):** 針對每一句的單字(word)加以分析，處理縮寫、日期、幣制、溫度等特殊符號。例如 “Add $20 to account 55734” 這句話經前處理後會變成”Add twenty dollars to account five five, three seven four.”。
3. **文字轉音素 (Text-to-phoneme conversion):** 將前一步驟所產生的單字逐字轉換為音素(phoneme)。例如”times”這個單字會被轉換成”t ay m s”這四個音素。
4. **聲韻學分析 (Prosody analysis):** 合成器解析整個句子的語意，決定唸這句話時的語調、快慢、斷句等等。
5. **輸出音檔 (Waveform production):** 根據 3,4 二個步驟所產生的音素與聲韻學資料，合成並輸出音檔。合成的方式有 Formant TTS 和 Concatenative TTS 二種方式，前者是使用人工的聲波合成，後者是使用人預錄的音檔處理接合而成，一般來說 Concatenative TTS 會產生較好的合成效果[27]。

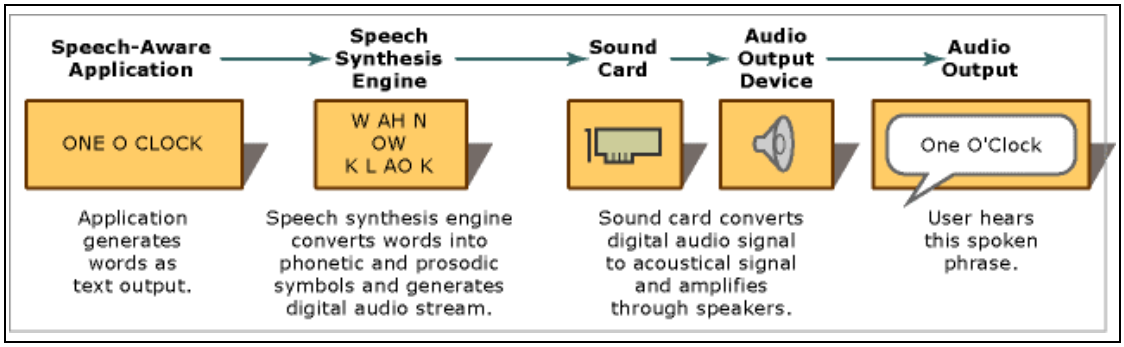

圖 2.1：語音合成的主要步驟 (資料來源[27])

2.3.2 語音辨識(Speech Recognition)

傳統 IVR(Interactive Voice-Response)中，使用者只能以電話按鍵輸入。在較先進的 ASR(Automatic Speech Recognition)系統中，可以使用語音辨識技術取代電話按鍵。電話按鍵表達能力有限，所以使用語音辨識可以讓使用者更方便輸入文字。在語音系統中使用的語音辨識引擎必須是 Speaker-Independent 且辨識率要相當高，才不會讓使用者感到不適。目前的語音辨識技術沒有辦法達到 100%的辨識率，所以比較重要的資訊 (如身份証字號、密碼或金額等)，仍須使用電話按鍵來做輸入動作。

語音辨識就是將人的語音轉成文字，在一般的應用中，須更進一步配合文法(Grammar)轉換成電腦可以理解的語意記號(Notations)。辨識的過程主要包含以下五個步驟:

1. **使用者輸入(User Input) :** User 從透過輸入界面說一個字或一句話，系統以類比方式捕捉其語音訊號(analog acoustic signal)。
2. **訊號轉換(Digitization):** 把捕捉到的類比訊號轉成數位訊號。
3. **切音(Phonetic Breakdown):** 訊號數位化後，將這個訊號切成音素。

4. **統計模型處理 (Statistical Modeling):** 利用統計模型，例如隱藏式馬可夫模型(HMM)，將這些音素對映到它們的 phonetic representation(類似音標的符號)。
5. **比對 (Matching):** 辨識引擎依據程式開發者寫的 Grammar 及步驟 4 所得到的 Phonetic representation 加上 Dictionary(phonetic representation 和 word 的對照表，如 thee → the)，來做比對，傳回一個可能辨識結果與辨識正確率信心指數(Confidence score)的配對列表(n-best list)。

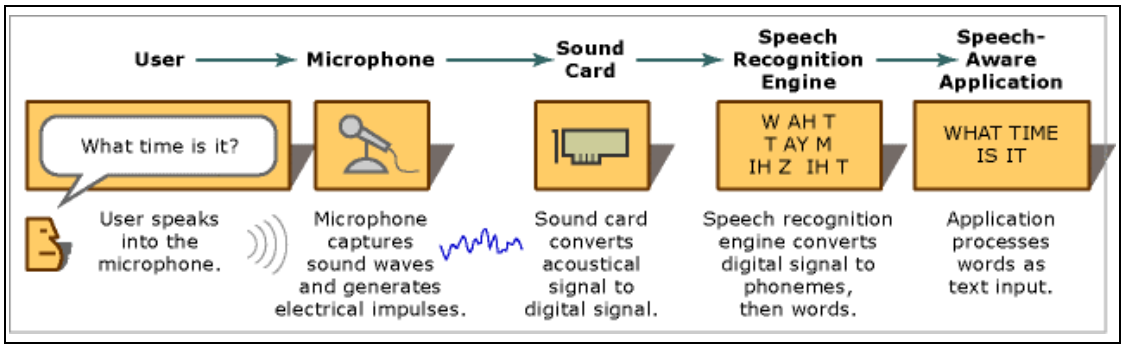

圖 2.2　語音辨識的主要步驟 (資料來源[27])

辨識引擎的狀態通常分為 Dictation 與 Rule Grammar 二種型式。Dictation 為聽寫狀態，辨識引擎會在所有的字彙中搜尋所有可能的辨識結果，通常辨識率較低。當能預期使用者說話的句型或可能的輸入文字時，可利用 Rule Grammar 來限制搜尋的範圍，這種型式可以大幅提高辨識的正確率。目前大部份的商用語音應用程式都會使用 Rule Grammar 限制搜尋的範圍。

一個功能完整的語音辨識引擎除了基本的辨識能力外，通常包含下列功能:

1. **插話機制 (Voice Barge-in):** 電腦語音在播放途中，使用者在系統未預期的情況下又說出一句話，此時系統中斷電腦語音的播放並辨識使用者的插話。
2. **Continuous/Discrete input:** 使用者語音輸入時要一個字一個字說，稱為 Discrete Input；使用者輸入時可連續說稱為 Continuous Input。一般的語音辨

識引擎都能同時接受 Continuous 及 Discrete 的輸入。

3. **Word spotting:**在 Continuous Input 時，抓 User 說一整句話時的關鍵字，以提高辨識率。
4. **Speaker Independent:**應該不只能辨識單一使用者的語音。

2.4 語音與虛擬環境的整合

具有語音界面的應用程式一般被認為開發難度較高。在每台電腦上，語音合成與辨識引擎只存在單一實體(Singleton)，通常會由所有該電腦上所執行的程式所共享。若架構未經適當的設計，很可能造成系統無法正常運作或效能低落。因此有一些學者針對如何寫作較好的語音應用程式，設計出可重用的語音應用程式框架(Framework) [43]。

虛擬環境與語音系統在執行時期均相當損耗資源，並且很有可能會獨佔某些 I/O 裝置使二者發生衝突(如有些虛擬環境會允許播放音樂)。舉例來說，若將二者的程式碼在同一個 JVM 中執行，會發生動畫停格或語音完全無法播放等問題。因此整個系統必須有適當的架構設計，並透過合宜的執行緒的控制，才能使二者達成一定的效能。

2.4.1 整合問題的種類

3D 虛擬環境與語音界面的整合的具體討論始於[35]，作者在 1995 年便對於虛擬環境與語音界面整合時會遇到的技術與設計上的問題詳細地加以分析與討論。他將虛擬環境與語音界面整合時會出現的問題分成三類*:Speech Recognition*、*Language Understanding* 及 *Interaction Metaphor*。

Speech Recognition 指的是執行語音辨識時，系統字彙(vocabulary)的多寡與辨識正確率的取捨(trade-off)的問題。系統的字彙愈大，所模擬出來的系統愈能夠接近自然對話的目標，但語意辨識的效果將愈差。故一般語音應用程式都會利用某些機制限制一定數量的字彙，以求達成良好的互動效果。其中最常見的是使用文法(Grammar)來描述及限制使用者可能的談話內容。

Language Understanding 指的是當系統正確地將談話化為文字內容後，如何理解這段文字的意義，根據所解讀的談話意義，決定下一段對話。

Interaction Metaphor 則為語音界面在虛擬環境中的定位。分成 Proxy、Divinity、Telekinesis 與 Interface Agent 四種。其中 Proxy 與 Interface Agent 是指人透過語音直接與許多中介的 agent 溝通，再由這些 agent 為我們改變虛擬世界狀態，而 Interface Agent 不同處在於它是以單一的 Agent 方式存在。Divinity 透過語音界面可以直接操作或改變虛擬世界狀態。Telekinesis 是人直接與虛擬世界中的物件溝通，由這些物件自行改變它們的狀態。

這個時期電腦的計算能力有限，語音辨識技術尚未和今日一樣成熟，故其系統並未進一步發展，虛擬環境與語音界面整合的藕合度也太高。而其對話內容必須植入程式中(Hard Coded)，沒有辦法達成動態對話的目的。McGlashan 似乎也注意到對話管理的重要性，他目前是 W3C VoiceXML[47]標準制定小組的領導人(Editor-in-Chief)。VoiceXML 可提供相當豐富的對話管理功能，目前已成為最被廣泛地接受與應用的 W3C 標準規格之一，而本研究其中一項目的即是以 VoiceXML 為基礎延展出適於多人虛擬環境下之對話管理機制。

2.4.2 整合問題相關研究

[12]選擇使用了 CSLU Toolkit 來做語音界面的整合，並實作了一個 GUI 界面的對話編輯工具，支援了基本的對話管理功能。但其語言使用 tcl/tk，且必須透過 CORBA 界面與即有的虛擬環境溝通。因此該論文作者也承認此系統在效能表現在並不理想。[50]針對這個問題，使用 Sun 所制定的 JSAPI 技術[28]對二者加以整合，但是其對話內容則儲存在資料庫中，沒有辦法達成動態的對話選擇與進階的對話控制。

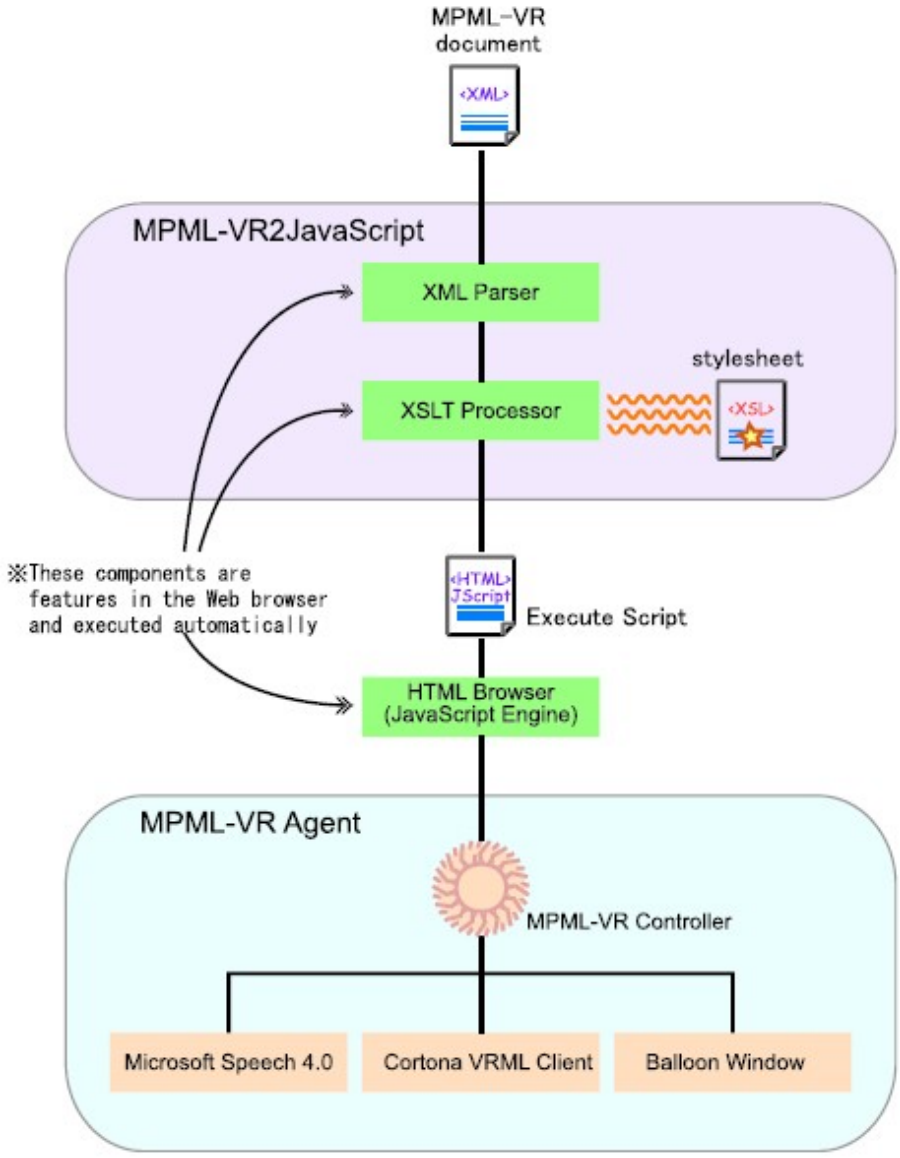

圖 2.3　MPML-VR 架構圖，本圖取自[51]

MPML-VR(Multimodal Presentation Markup Language for VRML)由 [51] 與 [14] 提出，主要是利用 XSLT 的技術，將 MPMP-VR 轉成 JavaScript 及 VRML 來達成動畫目的，如圖 2.3。此系統的核心是一個 MPML-VR Controller，負責操作 Microsoft Speech SDK, VRML Browser 及 Windows。

MPML-VR 是透過 Javascript 與 VRML 的 Node 溝通(如圖 2.4)。MPML-VR 基於原來的 MPML 機制加以擴充，訂定了一個新的 MPML-VR 標籤語言。透過 XSLT 的轉換來和原有的 VRML-JavaScript 機制對映，可以說是一個完全基於 VRML 的整合方式，其成果可以直接在 VRML Browser 中觀看。這種做法最大的問題在於一般的 EAI 只能夠和 Microsoft VM 溝通。而 Microsoft VM 只支援到 JDK 1.1，所以一些進階的 Java 機制都無法使用，限制了系統的擴充性。

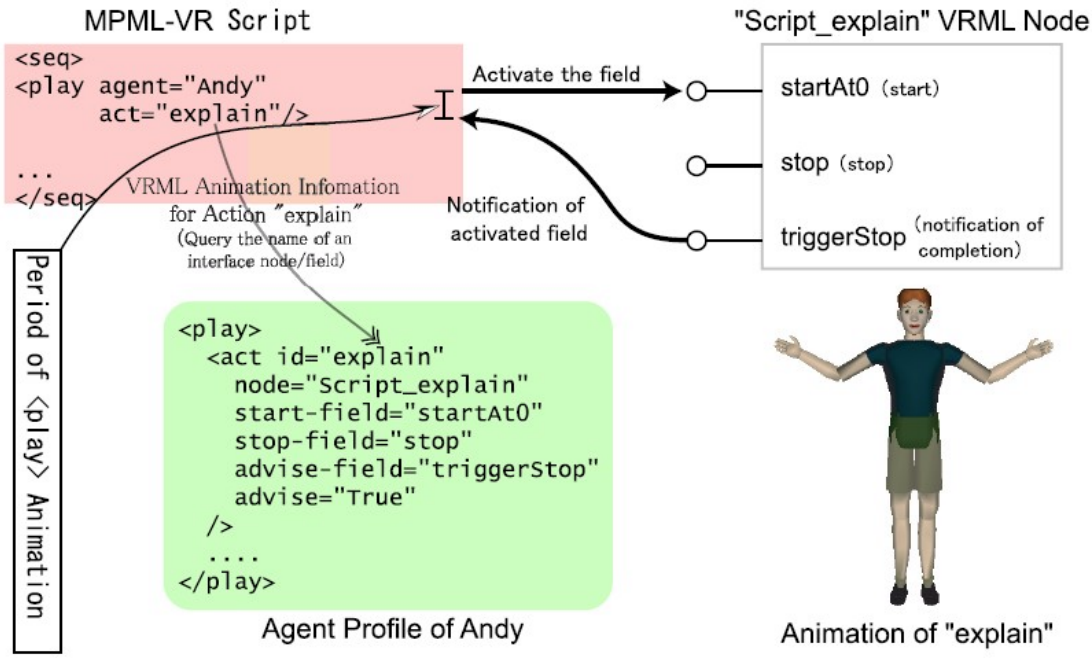

圖 2.4　MPML-VR 系統與 VRML Browser 的溝通，本圖取自[51]

根據我們的開發經驗，透過 VRML-JavaScript 來和語音系統互動及同步實作上相當困難，而在[51] 與[14]中並沒有交待 MPML-VR 的實作方式。

2.4.3 虛擬環境和語音界面整合時的考量

我們將上一節介紹的語音界面與 3D 虛擬環境整合方式加以分析整理，發現整合虛擬環境與語音系統時有以下幾個重要考量:

1. Client 端的選擇 :

使用者在客戶端必須透過某些機制才能監看到世界的狀態和其它虛擬物體進行互動。虛擬環境客戶端的實作一般分為二類，一種較簡易的方式是採用內嵌在一般 Internet Explorer 或 Netscape 中，以 Plug-in 方式存在的 VRML 瀏覽器。其與虛擬平台的溝通是以 Applet 的方式存在，並透過 EAI 界面與 VRML 瀏覽器互動，如[5]、[51]與 [14]。

這種方法的好處是使用了標準(VRML)來描述世界，相容性佳，且不需要自行開發虛擬環境瀏覽器。但其最大的缺點在於許多 VRML 瀏覽器都使用 ActiveX 技術來實作 EAI 界面，如此一來使用的瀏覽器品牌與 JDK 的版本都受到嚴重的限制。另外各家瀏覽器對 Plug-in 與 Applet 在多執行緒的實作方式與支援也不同，整合時容易發生鎖死或嚴重延遲的情況。上述工具大部份都未開放原始碼，所以也造成了除錯的困難。

另一種是在客戶端安裝並執行特定的客戶端(Client)程式，透過此客戶端程式，使用專有的協定與虛擬環境平台互動。這種方式是否適用最大的挑戰在於 3D 圖形顯示的處理與實作。目前 Java 3D 技術提供了強大的 3D 函式庫，而 X3D(VRML 的 XML 版本)也提供了相容於 Java 3D 的 API，因此即使是自行開發的客戶端程式，也可以很容易地支援標準的 VRML。所以我們認為整合虛擬環境與語音界面這二種異質系統時採用這種做法將具有較大的彈性，而自行實作的

程式也較不容易受到各種先天條件限制。

2. 語音引擎的選擇 :

語音引擎包含 ASR 與 TTS，在選擇時除了效能、辨識/語音合成效果之外，最大的考量可能是其提供的 API。因為這和其虛擬環境整合時的難易度息息相關。例如假設虛擬環境 Client 端是以 Java 寫作，則語音整合部份便可輕易地透過 JSAPI 來實作。若語音引擎完全是以 Java 語言實作，則整合更加容易，虛擬環境只要直接呼叫其類別的公開方法即可互相傳送訊息。目前以 Java 寫成的語音引擎有 FreeTTS[20]與 CMU Sphinx 4[42]。

此外根據加入語音界面的 interaction metaphor(請參見 2.4.1)的不同，會需要不同程度的文法支援。不同的語音引擎支援的文法規格及廣泛程度也有所不同。例如在事先可以確定預期語音輸入字彙範圍的應用中(如 Voice Command)，可選擇以 Rule Grammar 來提高語音辨識的正確率；若不能預期字彙範圍(如虛擬環境中人物的自由對話)，則應採用 Dictation 的方式來做語音辨識。

3. 語音介界如何與虛擬環境互通訊息:

虛擬環境與語音界面整合之後，二個模組間必須建立協調與同步的機制。訊息互通的機制和實作時所選擇的虛擬環境平台與語音引擎有密切的關係。例如[12]選擇使用了使用 tcl/tk 語言做為 API 的 CSLU 語音引擎，而其虛擬環境 VARC C6 必須透過 CORBA 界面與外界連接，故作者只能選擇以 CORBA 技術來做訊息互通的，也因此造成了嚴重的效能低落。因此訊息互通的機制應儘可能採用公開的標準，前後端最好也使用一致的技術。

由於虛擬環境與語音界面對系統資源都有獨佔的傾向，所以有的研究將語音界面與虛擬環境分成二個不同的 Process，中間再以 Socket 溝通，如[12]與[50]。此時 ASR 與 TTS 會以伺服器的方式存在，外界可以根據預先訂好的協定，透過 TCP Socket 要求 ASR 或 TTS 伺服器完成其語音辨識或合成功能。(如圖 2.5)

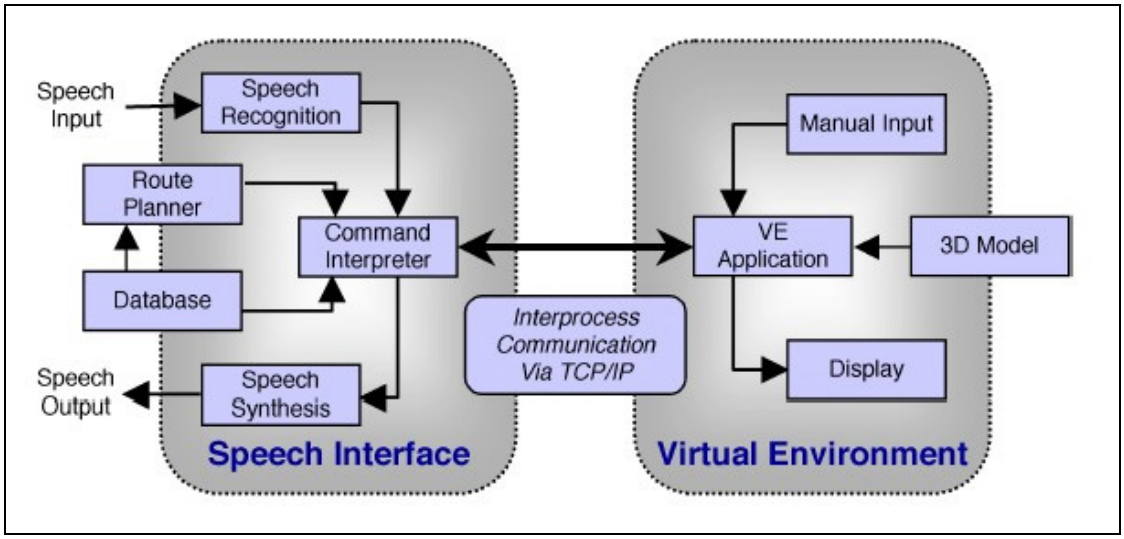

圖 2.5　透過 Socket 與虛擬環境整合，本圖取自[50]。

4. 對話管理的機制

根據我們與虛擬人物的對話，系統應能動態決定下一段對話為何。若系統不支援動態對話，則每一次談話的內容都不變，將大幅降低虛擬環境的真實性。對話管理的做法可分為二類，其中一種是將對話的內容存在資料庫，另一種則是使用腳本(scripting)的方式來達到動態產生對話的目的。這一部份的相關研究我們將在 2.5 節詳述。

2.5 VoiceXML 與互動式對話管理

2.5.1 VoiceXML 與 FIA (Form Interpretation Algorithm)

傳統互動式電話語音系統(IVR, Interactive Voice Response)牽涉到底層的電信技術與對話流程的控制，設計過程十分繁鎖。由 W3C 所制定的 VoiceXML 最主要的設計目標是希望將較簡易且較普及的 web-based 應用程式開發觀念用來取代傳統建構互動式電話語音系統的模式[47]。其基礎架構如圖 2.6 所示，Implementation Platform 表示使用者端的角色，Implementation Platform 透過 VoiceXML 解譯器(VoiceXML Interpreter)向文件伺服器(Document Server)要求對話文件(Dialog Document)。該文件會委由 VoiceXML 解譯器加以解讀並負責與使用者互動、取得回應，再據以向 Document Server 要求下一份對話文件。

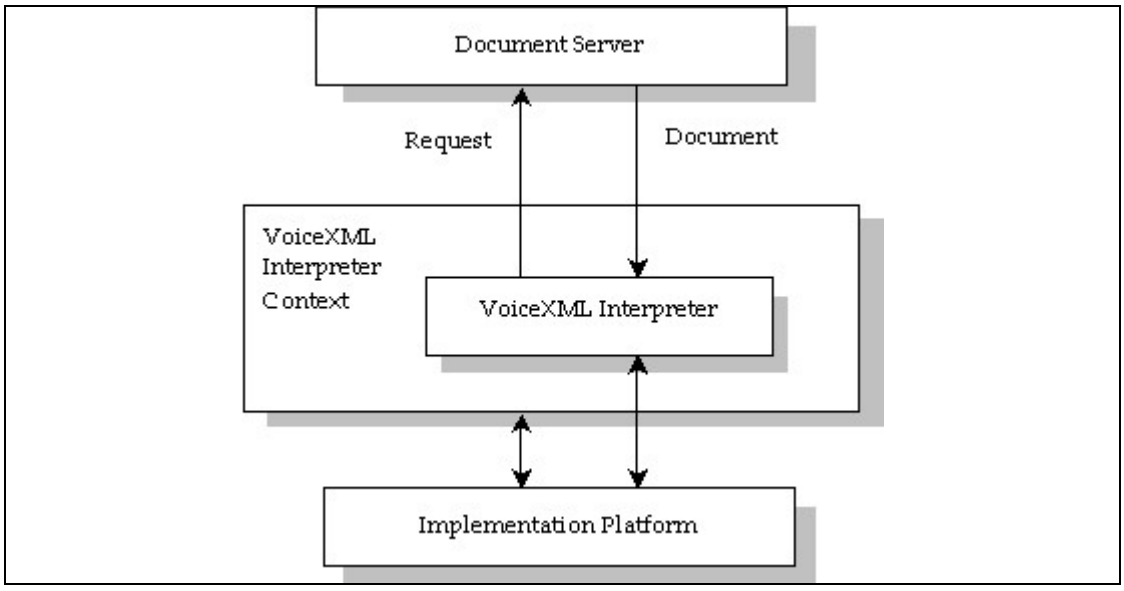

圖 2.6 W3C VoiceXML 基礎架構 (本圖取自[47])

在規格書中指出，Form 是 VoiceXML 中最關鍵的元件。VoiceXML 對話管理是使用 slot-filling 的方式來模擬一段對話。一份 VoiceXML 文件中可以包含一

個或多個 form 元素，而一個 form 元素則對映到一個對話的狀態(dialog state)。一個 form 元素是由許多 field 組成，代表系統希望使用者回答的問題。通常 VoiceXML 文件的作者可以在 field element 中指定文法(grammar)來限制使用者的字彙(vocabulary)，也可以指定 form level grammar 來達成初步的 mix-initiative 對話[40]。在一份文件中，預設的情況下，form 是依照前序被執行，一個 form 執行完畢之後，也可以透過 submit 或 goto 轉移到另一個 form 或 VoiceXML 文件。

VoiceXML 解譯器必須根據規格書中所提供之 FIA (Form Interpretation Algorithm)針對 Form 加以處理，根據其文意(semantic)來操作語音辨識和語音合成引擎來和使用者互動，並搜集使用者的回應。在主迴圈(Main Loop)會選擇合適的 Form，並加以執行。每次的迴圈都會經過三個步驟:Select Phase、Collect Phase 與 Process Phase，如圖 2.7 所示。在規格書[47]中列有詳細的步驟可供實作時參考。

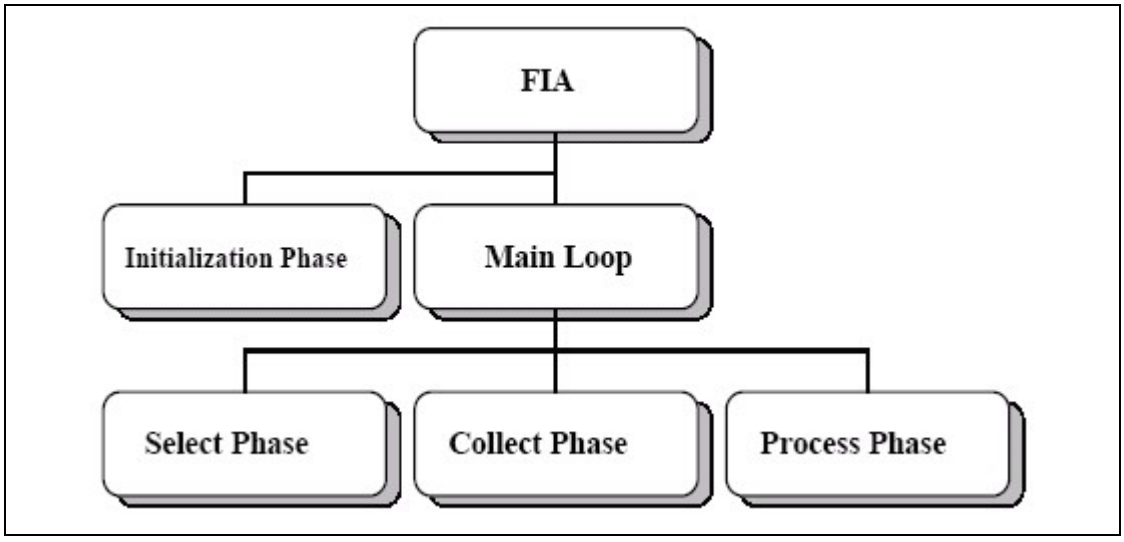

圖 2.7　Form Interpretation Algorithm，本圖取自[45]

2.5.2 對話控制機制相關研究

VoiceXML 的對話管理模型在許多研究中被廣為採用，*Galatea* [29] [41]是由日本東京大學知識情報工學系 (Univ of Tokyo Graduate School of Information Science and Technology) 所發展的一個可模組化，具擴充性的 ASDA (Anthropomorphic spoken dialog agent)平台，GalateaDM 是此平台的其中一個模組，用來做對話控制(Dialog Control)。他們以 VoiceXML 當作 Galatea 平台的對話控制的基礎，再加上一些擴充功能來達成和 Galatea 互動的目的。

例如圖 2.8 這一段敘述中，<block>及<prompt>原本是 VoiceXML 中標準的 Tag，<log>及<native>則是屬於擴充的機制，我們可以觀察到<native>...</native>中其實內嵌了類似動畫腳本的敘述。

```
<block>                          // 在 VoiceXML 中，block 代表「要被執行的單位」
    <log>greeting begin</log>    // 記錄系統訊息
    <native>to @FSM set HedRotAbs.1 = 0 10 0</native>   // 臉部動畫
    <prompt>Hello!</prompt>      // 語音合成
    <native>to @FSM set HedRotAbs.1 = 0 0 0</native>    // 臉部動畫
    <log>greeting end</log>      // 記錄系統訊息
</block>
```

圖 2.8　Galatea 的對話機制

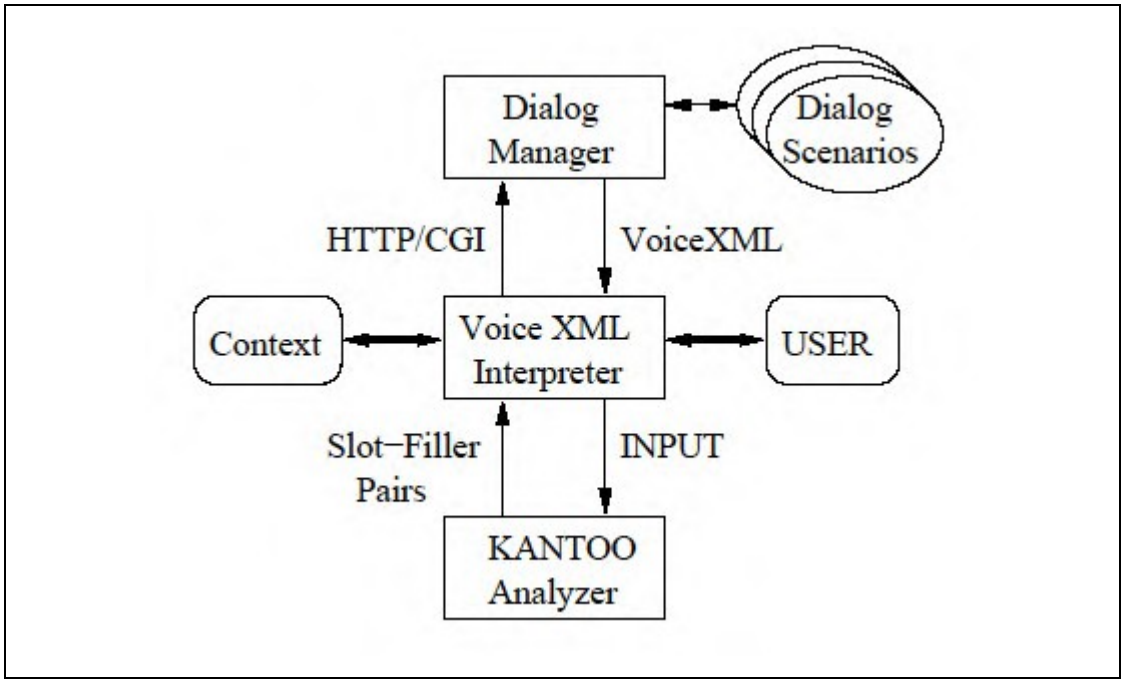

圖 2.9　DialogXML 系統架構，取自[38]

另外在[38]中，作者則認為 VoiceXML 的 form-filling 機制沒有辦法輕易地表達出複雜的 state transition，以建立進階的對話系統，所以提出了 DialogXML 來補強 VoiceXML 的功能。DialogXML 的語法完全是由作者自行訂定，而非由 VoiceXML 的語法來擴充。其架構如圖 2.9 所示，DialogXML 由 Dialog Manager 元件依照對話場景(Dialog Scenarios)動態地產生出 VoiceXML，再交由 VoiceXML Interpreter 處理。DialogXML 將整個對話系統看成一個 state diagram，語法則以表達 state 與 arc 為中心。在執行時此系統透過轉換機制(Dialog Manager)將 DialogXML 轉換成 VoiceXML，才交由 VoiceXML Intepreter 來執行(如圖 2.10 所示)。舉例來說，在下面這一段 DialogXML 的例子中，包括了二個 state(s1 , s0)，及一個 arc(連結 s1 及 s0)。當 s0 透過 arc 進節 s1 時，系統就會播放一段音檔"How can I help you"。

```
<DialogXML>
  <dialog name="main">
    <state name="s0">
      <arc>
        <dest state="s1"/>
      </arc>
    </state>
    <state name="s1">
      <action><prompt>How can I help you?</prompt></action>
      <arc><dest state="s0"/></arc>
    </state>
  </dialog>
</DialogXML>
```

圖 2.10　DialogXML 腳本，取自[38]

所以 DialogXML 其實不是 VoiceXML 語法的擴充，而是在 VoiceXML 之上，而加上一層更高階的對話腳本語言。我們認為對於對話腳本語言的寫作者來說，將對話過程以 State Diagram 來表達，其實不一定是一種簡化，若加上一個合適的 Authoring 工具，才可能真正簡作對話的定義流程。

DialogXML 的另外一個問題是，取得 DialogXML 腳本後，系統必須先將腳本轉換成 VoiceXML，再交由 Interpreter 來處理。如此一來在多人環境中運作時效能將會是一個很大的問題。另外 DialogXML 也沒有考量多人對話時可能會引起的同步問題。

2.6　軟體樣式(Software Patterns)

樣式(Pattern)一般指針對時常發生的同一個問題，大家所經常採行的解決方案。以軟體設計工作來說，每一個有經驗的工程師建構程式時可能都會發展出自己的 Patterns。例如:每個系統時常都會用到登入功能，當我們對登入功能有一定實作

經驗後，會發現要設計一個良好的登入功能都有一定法則，於是便形成了一個可能的設計樣式。

學者一般習慣將得出的樣式記錄在樣式目錄(Pattern Catalog)中。這種做法最早是在建築學領域被採用。1977 年 U.C. Berkeley 的教授 Christopher Alexander 出版了一本建築學用的樣式目錄[2]。在 1995 年 Eric Gamma 等人出版了 Design patterns: Elements of Reusable Object-Oriented Software[21]，首先將 23 個設計樣式(Design Patterns)以樣式目錄的形式記錄下來。1996 年 Frank Buschmann 等人出版了 Pattern-Oriented Software Architecture, Volume 1:A System of Patterns [9]，提出了軟體樣式系統理論(A System of Patterns)，從此之後軟體樣式研究便開始受到普遍的重視。

在[9]所提出的軟體樣式分類中，由「鉅觀」到「微觀」可以分成下面三類:

1. **Architectural Patterns** :通常指軟體架構上概念性的樣式，其實作通常很難單純以少數幾張 UML 的 Class Diagram 與 Sequence Diagram 來表達。例如:MVC Pattern、Layered Pattern。如[18]與[3]都屬於這方面的研究。

2. **Design Patterns**:指設計類別時用來完成特定需求的解決方案，雖然也是屬於概念層級的重用，但比較容易以數張 Class Diagram 與 Sequence Diagram 來表達。例如:[21]中的 23 個 patterns。

3. **Idioms**:最低層級的軟體樣式，一般指程式設計實作時的建議做法，例如在如[8]中建議我們使用 Java 時，處理大量接合字串時一般用 StringBuffer 取代 String；覆寫 equals()方法時也同時必須覆寫 hashcode()方法。

軟體樣式在資訊科學領域上的應用非常廣泛;目前為止最成功的是在企業應用領域，如[18]與[3]。也有人將樣式的概念應用在軟體工程的領域上，如應用在軟體開發流程的 Process Pattern[4]，用來開發 Framework 的開發流程樣式[10]及應用在 Configuration Management 的 SCM 樣式[6]等。另外在系統需求分析有[17]提出來 Analysis Pattern，在解決多執行緒問題也有 Concurrency Pattern[31]。本研究將嘗試以軟體樣式的概念應用在 XAML-V 平台的實作過程。