第四章

互動式語音界面之對話管理腳本語言 - XAML-V

4.1 XAML 動畫腳本語言

3D 虛擬環境中虛擬人物的動作，大多以動作抓取等離線方式錄製後，再以罐裝動作的方式播放，產生過程繁瑣且耗時。在[33]中提出了一套動畫腳本語言，稱為 eXtensible Animation Markup Language(XAML)，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。

下圖是一個 XAML 的基本範例:

```
<AnimItem DEF =”WaveWalk” cycle=”2000”>
  <AnimImport src=”Walk”>
  <AnimItem DEF=”SimpleWave” cycle=”1000”>
      <Node target=”r_shoulder”>
          <OrientationInterpolator   key =”…” keyValue=”…” />
      </Node>
  <AnimItem>
</AnimItem>
```

圖 4.1　XAML 動畫描述語言的範例

在任何 XAML 腳本中，都是以<AnimItem>為根元素。新的動畫內容指定可以分為兩個部分：屬性(Attributes)和內容(Content)。屬性主要是管理邏輯上的控

制，例如撥放時間、動畫內容篩選等來控制新動畫的表現。在內容的指定上，可在<AnimItem>中則可以放入要新產生的動畫實體，動畫實體內容可以藉由<AnimImport>作動畫元件重複利用或者是利用<Node>作低階的動畫指定。XAML 腳本語言的語法結構如圖 4.2 所示:

```
<AnimItem> := (<AnimItem> | <AnimHigh> | <AnimImport> | <AnimTransition> )* | <Node>*
<AnimHigh> := text
<AnimTransition> := (<AnimItem> | <AnimImport>)*
<Node> := (<OrientationInterpolator> | <PositionInterpolator>)*
```

圖 4.2　XAML 腳本語言的語法結構

XAML 將動畫分為三種指定方式：高階、中階和低階。在低階的表示法中，首先要在 Node 的 target 參數中指定要變化的物體。例如，動畫的目標如果是虛擬人物的人體動作，則 target 給定的字串可以是 H-Anim [24] 中所定義的各個關節點(Node)的名稱。

在低階動畫中，物體移動轉動都是參照所給定的 key frame mapping，這部分是沿用 X3D [X3D] 的動畫指定方式，分為<OrientationInterpolator>和<PositionIntepolator>兩種，該標籤包含 key 和 key value 二個屬性。

高階的動畫指定使用口語文字，例如“Walk to café.”。在目前的 XAML 實作中只接受簡單的文法結構，也就是動詞+介詞+名詞，而且動詞及受詞也必須在動畫資料庫中有紀錄的才可以被接受。

除了可以達成不同精細程度的動畫控制之外，XAML 也可以使用「元件內嵌」的方式被延展[32]。XAML 定義了一個<AnimPlugin>的標籤，所有在此標籤

內的元件都會被視為 Plug-in 元件加以處理。藉著這個功能，開發人員可以在 XAML 平台之上，自行實作新的 Plug-in 元件，並針對 XAML 腳本語言加以擴充。我們在下一節中將介紹的 XAML-V(XAML - Voice Extension)腳本語言，即為以 XAML 平台 Plug-in 的方式，針對 XAML 腳本語言的語音辨識、語音合成及對話管理功能加以擴充。

4.2 XAML-V(XAML Voice Extension)概觀

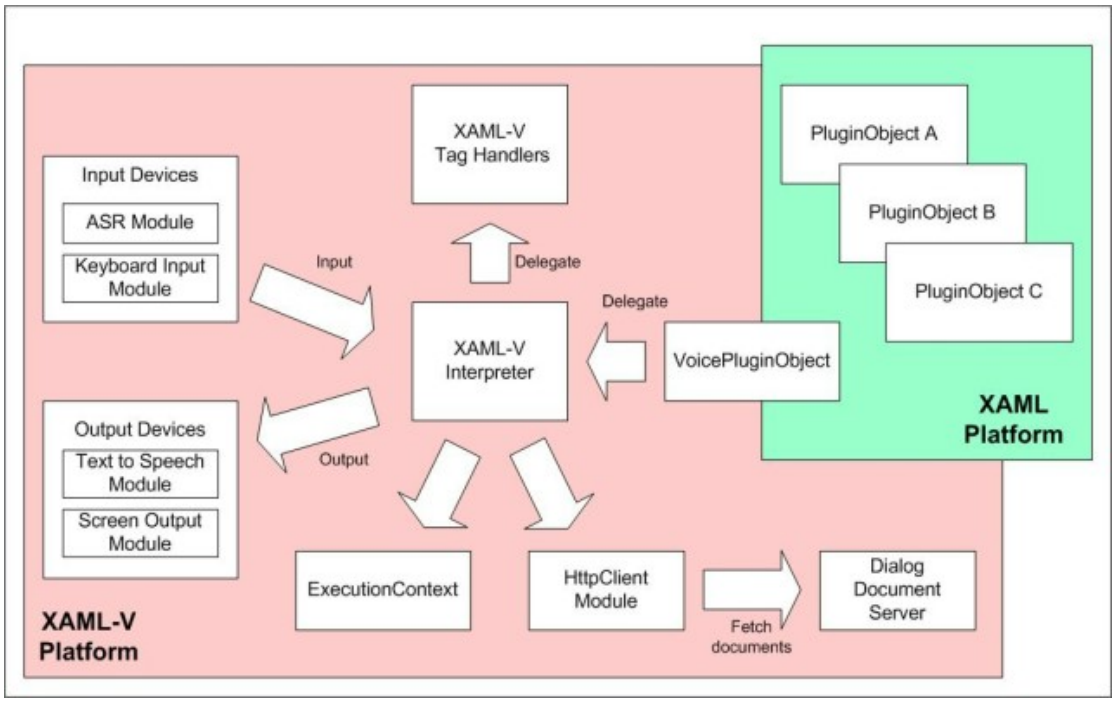

圖 4.3 XAML-V 平台

XAML-V(eXtensible Animation Markup Language - Voice Extension)是由 VoiceXML 語言擴充而成，以 Plug-in 的方式加入 XAML 平台，讓原本只能播放動畫腳本(Animation Script)的 XAML 平台具有播放對話腳本(Dialog Script)的功能。如圖 4.3 所示，當 XAML 平台發現 XAML-V 的腳本語言時，會將這一部份的腳本委派(delegate)給 VoicePluginObject 處理，經由 XAML-V 解譯器(XAML-V

Interpreter)，使用者可透過 TTS(Text to Speech)與 ASR(Automatic Speech Recognition)等語音模組與虛擬環境中其它虛擬人物溝通，達成互動的目的。

XAML-V 針對 XAML 所缺乏的語音辨識、語音合成及對話管理功能加以擴充，以 W3C VoiceXML 2.0 為基礎發展而成。XAML-V 同時繼承了 VoiceXML Form 型態的互動式對話管理特色與 XAML 的動畫控制功能。藉由這二項功能的整合，可以達到在虛擬環境中透過語音界面與其它虛擬人物互動的目的。

XAML-V 繼承了 VoiceXML 可在 Server 端動態產生對話的特色[11]。對話執行過程中，XAML-V Interpreter 可依照 User 的回應，向對話文件伺服器(Document Server)要求不同的文件以產生下一個 XAML-V 腳本文件。透過 Server Side Scripting(如 JSP/Servlet 或 PHP)技術，Document Server 也可以存取其它外部資訊來源(如:資料庫、Web Services 等)，依據各種不同情況，動態產生不同 XAML-V 腳本文件。

VoiceXML 原本是針對電話系統而設計，而 Telephony、Call Control 等與電話系統相關功能在虛擬環境並沒有相對應的需求，故未納入 XAML-V 腳本語言，因此 XAML-V 僅實作了 W3C VoiceXML 2.0 的子集合。但另一方面來看，XAML-V 為了適應多人虛擬環境的特色，也對原有的 VoiceXML 做了擴充。這些語法的擴充主要可以分成四個部份：動畫整合機制、對話協議機制、對話廣播機制與對話腳本擷取的代理。以下我們分別就各個機制簡單加以說明：

4.2.1 動畫整合機制

為了與動畫功能同步並加以協調，在 XAML-V 中我們增加了一個 <animation>標籤，使得對話腳本語的作者在 XAML-V 中也能指定相對應的動畫

並加以播放。目前在 XAML-V 中能支援 Form 層級(Form Level Animation)與 Field 層級(Field Level Animation)二種不同的精細程度的動畫控制。具體的語法與運作方式我們在 4.3 節有詳細的說明。

4.2.2 對話協議(Dialog Negotiation)機制

在第三章曾討論過進行對話與傳播對話時 Client 端之間必須傳送一連串的腳本訊息來做溝通。因此我們在 XAML-V 中加入一個新的<protocol>元素來表達這些多人環境下的對話協議(dialog negotiation)機制。包括了「要求對話」、「接受對話」、「拒絕對話」、「確認對話」與「結束對話」等五種模式(context)。具體的語法與運作方式我們在 4.4 節有詳細的說明。

4.2.3 對話廣播機制

在 3.1 節中我們將單純廣播，在 Client 直接播放放出來，且沒有互動的 Script 定義為 *Broadcasting Script*；而會詢問某些問題，並取得使用者回應的腳本稱為 *Dialog Script*。為了在 XAML-V 實作此一機制，我們將 XAML-V 中的 form 新增一個 xml attribute 稱為 type，其值可能為 dialog 或 broadcasting，分別對應到 *Dialog Script* 與 *Broadcasting Script*。所以依據此一屬性，XAML-V 解譯器得以判斷此一腳本的種類並做不同的處理。

4.2.4 對話腳本擷取的代理(Proxy Request)

使用 VoiceXML 對話架構進行對話時必須有「對話文件伺服器(Document Server)」的輔助，以便根據 Client 所送來的回應，再動態地回應下一段對話腳本(Dialog Script)。當我們將這種 VoiceXML 的對話機制運用到多人虛擬環境中二個虛擬人

物的對話，並非十分直覺。因此在 3.2.1 節中我們針對原有的 VoiceXML 模型加以改良，讓多人環境中的對話腳本傳播較為自然。

XAML-V 以<proxy-request>語法來實現此一構想。Us 會將執行對話腳本所取得的使用者回應，打包成<proxy-request>訊息，再將此訊息送至 Ss，由 Ss 根據此訊息的內容向 Document Server 下達 Http Request 取得下一段對話腳本，並回傳給 Us。具體的語法與運作方式將在 4.5 節做詳細的說明。

4.3 XAML-V 的對話管理與動畫整合

目前有許多成功的動畫整合與對話管理研究都是基於 VoiceXML 的對話模型擴充而來。例如東京大學的 Galatea 系統[29] [37]是擴充 VoiceXML 來支援其動畫平台。這套系統在臉部動畫及其表情表現相當優秀，但由於 VoiceXML 的設計之初是以在電話中進行的語音應用為考量，如要在虛擬環境中運用，就必須改寫或擴充 VoiceXML。所以[29] [37]的研究也是以 VoiceXML 為主，加上 Animation 的 Tags 來擴充。

但是這種做法很可能變成一種反客為主的做法。在虛擬環境中，動畫的功能應該是主體。本研究中我們採取以動畫腳本為主的做法。雖然在語音與動畫整合方面我們採用和 Galatea 相似的設計，取 VoiceXML 用於對話管理的子集合來擴充 Animation Tag。但在 XAML-V 平台之上仍然以動畫腳本(XAML)為主體，而將 XAML-V 視為 XAML 的擴充語言。

Galatea 的另一項嚴重限制是其對話文件解譯器(Interpreter)並不能和對話文件伺服器(Document Server)溝通。在 XAML-V 平台中，XAML-V 解譯器內含 Http

Client 模組，文件解譯器可再向對話文件伺服器下一份文件，進而和 Server Side Scripting 技術結合，達成 Server Side Dialog Control 的功能。

在對話管理方面，以原始的 VoiceXML 對話機制若要做到 Client Side Dialog Control，需要搭配相當複雜的 Java Script。所以有些 Client Side Dialog Management 研究針對 VoiceXML 的對話模型進行改良[38]，自動產生這些複雜的 Java Script。[38]批評 VoiceXML 缺乏進階的 Dialog Control 機制，不過他們可能忽略了 VoiceXML 也可以利用 Server Side Scripting 來做對話控制[11]。就像目前的 Web Programming 一樣，Client Side 的 JavaScript 和 Server Side 的 JSP/ASP 可以合作得很好，但不能完全捨棄另一方。目前 XAML-V 的實作是採用 Server Side Scripting 來控制對話的機制。

圖 4.4 是一個典型的 XAML-V 對話腳本，指示 Client 端播出一段”No Thanks” 的語音。由於 XAML-V 是基於 XAML 動畫腳本語言擴充而來，所以所有的 XAML-V 都必須位於<AnimPlugin>的元素之內，XAML 解譯器看到此元素之後，就會將控制權轉交給 XAML-V 解譯器繼續執行。

```
<?xml version="1.0" encoding="UTF-8"?>
<AnimItem>
    <AnimPlugin>
        <xaml-v version="1.0">
            <block>
                <prompt>No,Thanks</prompt>
            </block>
        </xaml-v>
    </AnimPlugin>
</AnimItem>
```

圖 4.4　XAML-V 對話腳本

圖 4.5 的 XAML-V 腳本則展示了動畫的整合功能。這段腳本首先要求 Client 播放二段語音，緊接者再播放一段高階動畫<AnimImport src=”listen”>，利用

XAML-V 所提供的<animation>元素，在 XAML-V 中又可以嵌入 XAML 動畫語言，當 XAML-V 解譯器遇到這種情況時，會將播此段動畫的要求轉交給 XAML 解譯器。目前 XAML-V 只可以支援單層的內嵌，也就是說，<animation>元素中的 XAML 不能再度包含任何的<AnimPlugin>。

```
<?xml version="1.0" encoding="UTF-8"?>
<AnimItem DEF="Services" playMode="seq">
 <AnimHigh>A Walk to B.location</AnimHigh>
 <AnimPlugin>
   <xaml-v version="1.0">
     <block>
       <prompt>Good Morning ,Sir,May I help you?</prompt>
       <prompt>You can say IMLAB,Computer Center, or No Thanks.</prompt>
          <animation>
            <AnimImport src="listen"/>
          </animation>
     </block>
   </xaml-v>
 </AnimPlugin>
</AnimItem>
```

圖 4.5　XAML-V 的動畫整合功能

含有 Form 元素的 XAML-V 具有和使用者互動的功能，而在 Form 元素中也可以內嵌適當的動畫，表達該對話時人物應有的動作。animation 元素可以直接成為在 Form 的子元素，如圖 4.6 所示，在講完第一句話”Good morning, Sir.”後，便開始播放”Stand”這段動畫，我們將這種動畫方式稱為 Form-Level Animation。另外，為了表達「問某個問題時的動作」，animation 元素也可以成為 field 的子元素，例如在圖 4.6 中，在 field 元素下有 prompt 與 animation 二個子元素，如此就可以指示 XAML-V 解譯器在播完” May I Help You? You can say : I.M. Lab, Computer Center or no thanks“後，就播出”Listen”的動畫，我們將這種方式稱為 Field-Level Animation。

```
<xaml-v version="1.0">
    <form id="helloForm" type="dialog">
        <prompt>Good morning, Sir.</prompt>
        <animation>
            <AnimItem dur="3000">
                <AnimImport src="Stand"/>
            </AnimItem>
        </animation>
        <field name="helpType">
            <prompt>May I Help You? You can say : "I.M. Lab", "Computer Center" or "no thanks".</prompt>
            <animation>
                <AnimItem dur="3000">
                    <AnimImport src="Listen"/>
                </AnimItem>
            </animation>
        </field>
        <submit next="helpFormResponse.jsp" namelist="helpType"/>
    </form>
</xaml-v>
```

圖 4.6　具有互動功能的 XAML-V

4.4　XAML-V 在多人虛擬環境下的對話協議機制

在 3.2 節我們討論了多人虛擬環境下如何啟始對話、進行對話與結束對話，在本章中我們將說明如何使用 XAML-V 來實現此一機制。

4.4.1　Client 端與 Server 端間傳送之訊息格式

IMNet 虛擬環境平台包含了 Client 與 Server 二種類型的程式，Client 端程式之間會透過 Server 互相繞送各種訊息，Server 另外還負責記錄各個 Client 的資訊，過濾並轉送訊息。

IMNet 虛擬環境平台統一定義了 Client 與 Server 傳送訊息的規格。如圖 4.7 所示，對 Server 來說，所有的訊息都只包含<imnet>元素。 Server 收到此一訊息後，根據其中的 from、to 等資訊，將訊息送給指定的目標。需特別注意的是，有時候我們會希望該訊息不要傳給某個目標，因此在傳送前，server 會將 except

中指定的目標去除。在圖 4.7 中，該訊息會傳送給 avatar_01 之外的所有 avatars。

```
<imnet from="avatar_01" to="all" except="avatar_01">
    <AnimItem>
        <!-- 包含實際動畫腳本資訊 -->
    </AnimItem>
</imnet>
```

圖 4.7　XAML 虛擬環境平台中傳送的訊息格式

XAML-V 以 Plugin 方式延伸 XAML 語言，所以所有 XAML-V 的元素都必須包含在在<AnimPlugin>元素中。傳送前會由 IMNet Client 將整個訊息打包，再經由 IMNet Server 繞送到特定目標。圖 4.8 展示了一個由 IMNet Client 打包完成準備傳送的 XAML-V 腳本格式。IMNet Server 解讀完這個訊息後，會將此訊息傳送給 target 屬性中所指定，名為 avatar_1 的 Client。

```
<imnet from="avatar_2" to="avatar_1">
    <AnimItem>
        <AnimPlugin>
            <xaml-v>
                <!-- XAML-V 腳本 -->
            </xaml-v>
        </AnimPlugin>
    </AnimItem>
</imnet>
```

圖 4.8　內嵌在傳送訊息中的 XAML-V 腳本語言

4.4.2　XAML-V 中的對話協議機制

根據 4.2 中所討論的啟始對話、進行對話與結束對話機制，我們在 XAML-V 中新增了一個<protocol>元素來表示對話未開始前二方的對話協議。<protocol>是在對話開始之前做對話協議的機制，不需要播放給使用者看。所以設計上應該在 Plug-in 組件中就被分開處理。換句話說，若一段對話腳本中包含了<protocol>元

素，則它只會在 Plugin 元件中被處理，不會被轉送到 XAML-V 解譯器執行。

無論是使用者或系統端的 Client 端都有一致的對話狀態(Dialog State)。請參考圖 3.7，在系統啟始時，所有的對話狀態需設都是「Not In Dialog」。以下我們舉例說明一個完整的對話過程與訊息。假設參與對話的人有一個人是使用者(Us)、一個人是系統(Ss)，且目前二個人都是在「Not In Dialog」狀態，現在使用者(Us)想要與系統啟始一個對話(Ss)。

1. Us 會先向 Ss 發出一段要求對話的訊息(Send Dialog Request)，根據圖 3.7，此時 Us 的 Client 會進入「Dialog Negotiation」的狀態。Us 所發出的訊息如圖 4.9 所示(為了讓 xml 碼清淅起見，在以下的討論中我們僅列出 XAML-V 腳本部份，這個 XAML-V 腳本會遵循 4.4.1 所描述的方式被 IMNet Client 打包並送出)，在<protocol>中我們使用了一個 dialog-negotiate 元素來表達對話的協議，source 屬性表示訊息的發出人，context 屬性表達訊息的類型。

```
<xaml-v>
    <protocol>
        <dialog-negotiate source="Us" context="request"/>
    </protocol>
</xaml-v>
```

圖 4.9　Us 送給 Ss 要求對話的訊息

2. Ss 收到 Us 送來的訊息(圖 4.9)之後，根據圖 3.7，「Receive Dialog Request」會使 Ss 進入「Dialog Negotiation」的狀態。此時如果發現自己並沒有在和別人進行對話，會發出一個同意對話的訊息(如圖 4.10)，在<protocol>中我們也是使用一個 dialog-negotiate 元素來表達對話的協議，context 的內容是「同意對話」(accept)。在多人環境中，多個 Clients 同時搶發訊息可能會造成同步的錯誤，避免這種錯誤的機制必須在 IMNet Client 端實作並處理(請參閱 3.2.4

的討論)。

```
<xaml-v>
    <protocol>
        <dialog-negotiate source="Ss" context="accept"/>
    </protocol>
</xaml-v>
```

圖 4.10　Ss 回傳給 Us 同意對話的訊息

3. Us 收到 Ss 送來的接受對話訊息(圖 4.10)後，根據圖 3.7，「Receive Dialog Accept」會使得 Us 進入「In Dialog」狀態，所以此時 Us 會將自己的對話權利(Dialog Lock)鎖住，並回傳一個對話確認的訊息(如圖 4.11)。

```
<xaml-v>
    <protocol>
        <dialog-negotiate source="Us" context="dialogAck"/>
    </protocol>
</xaml-v>
```

圖 4.11　Us 回傳給 Ss 確認對話的訊息

4. Ss 收到 Us 送來的對話確認訊息(圖 4.11)後，根據圖 3.7，「Receive Dialog Ack」會使得 Ss 進入「In Dialog」狀態，所以此時 Ss 也會將自己的對話權利(Dialog Lock)鎖住，再透過 HTTP 協定向對話文件伺服器取得第一份對話腳本，回傳給 Us。圖 4.12 即是一個假想的對話腳本，由 Ss 回傳給 Us。於是 Ss 與 Us 此時便進入了對話狀態。

```
<xaml-v>
    <form id="helloForm">
        <block>Good Morning ,Sir,May I help you?</block>
        <field name="helpType">
            <prompt>You can say IMLAB,Computer Center, or No Thanks.</prompt>
            <animation>
                <AnimImport src="listen"/>
            </animation>
        </field>
        <block>
            <submit next="next.jsp"/>
        </block>
    </form>
</xaml-v>
```

圖 4.12　Ss 回傳給 Us 第一份對話腳本

5. 此時 Ss 同時也要準備一份如圖 4.13 的 Broadcasting Script 向虛擬環境中的其它 Clients 廣播。開始播放此動畫之後虛擬環境中的其它 Clients 便能看到 Us 與 Ss 對話的情境與語音。值得注意的是在多人環境下，同時會有多個 avatar 存在，因此<AnimItem>元素中多了一個 model 屬性指定動畫播放的對象。另外由於是廣播用的腳本，所以 form 的 type 也會被設為 broadcasting。

```
<xaml-v>
    <form type="broadcasting">
        <prompt>Good Morning ,Sir,May I help you?</prompt>
        <prompt>You can say IMLAB,Computer Center, or No Thanks.</prompt>
        <animation>
            <AnimItem model="avatar_1">
                <AnimImport src="listen"/>
            </AnimItem>
        </animation>
    </form>
</xaml-v>
```

圖 4.13　Ss 的 Broadcasting Script

6. 假設此時另一個使用者(Ut)想要向系統(Ss)要求對話，因此向 Ss 發出了一個對話的要求。此時 Ss 檢查自己的對話權利(Dialog Lock)是鎖住的，因此回傳如圖 4.14 的拒絕訊息。Ut 的 Client 端程式收到此訊息後會回傳給使用者適當

的錯誤訊息。

```
<xaml-v>
    <protocol>
        <dialog-negotiate source="Ss" context="reject"/>
    </protocol>
</xaml-v>
```

圖 4.14　Ss 回傳給 Ut 的拒絕對話訊息

7. 最後 Us 希望和 Ss 中止交談，於是向 Ss 送出一個 endDialog 的訊息(如圖 4.15)，根據圖 3.10，此時 Us 會回到「Not In Dialog」狀態，並解除自己的 Dialog Lock，而 Ss 收到之後根據圖 3.7 也會自動解除自己的 Dialog Lock，並回復到「Not In Dialog」狀態。在協定的設計上，任何一方皆可提出中止對話的要求，提出後雙方馬上會中止正在進行的對話。

```
<xaml-v>
    <protocol>
        <dialog-negotiate source="Us" context="endDialog"/>
    </protocol>
</xaml-v>
```

圖 4.15　Us 傳送一個中止交談的訊息給 Ss

4.5　XAML-V 對話腳本擷取代理 (Proxy Request)

虛擬環境中的 Avatar 可能以二種形式存在，一種是由使用者透過 Client 端介面控制的 Avatar，另一種則由系統直接控制的 Avatar。但對於虛擬環境中的人物來說，這二者並無差別，在理想情況下，一個虛擬環境系統的使用者在和另一個使用者交談時，應該無法區分與他交談的人是真正的使用者或是由系統來扮演。

在一方是系統而另一方是人類扮演的交談過程中，XAML-V 腳本所扮演的角色是指示系統 Avatar 如何與人類進行對話，以及人類給予回應後該有什麼行為，或是講什麼話。我們在 3.2.1 節中討論過，VoiceXML 的對話模型原本是專為雙人對話所設計，擴充至多人環境時，會有不自然的情況發生(請參考圖 3.4)。

在改良後的新模型中(請參考圖 3.5)，對話腳本執行的結果會經由 IMNet Server 回傳給 Ss，再委由 Ss 代為擷取下一段腳本。原本應由 Us 透過 Http Request 取得的腳本都會委由 Ss 來取得，與一般 Http Proxy Server 的運作方式極為類似，因此我們稱這種方式為 ProxyRequest。

這種對話模型有下列三項優點:

1. 對 Us 來說，並不了解 Ss 是如何產生 Dialog Script，所以可以達到所有虛擬人物都*一視同仁的效果*，不需要刻意對系統 Avatar 與人扮演的 Avatar 加以區分。
2. 讓*所有的 Avatar 在二種模式間自由切換*。舉例來說，當使用者不方便操作 Avatar 時，可以將 Avatar 委由系統管理。此時若別人與他對話，系統會自動根據所抓取之對話腳本與他人進行對話。
3. 所有的資訊皆經過 Ss 傳送，所以 *Ss 有機會對 Us 的回應資訊做前處理，而 Us 也有機會隨時掌握 Ss 的情況*。例如在原先的模型中(圖 4.17)。建立對話後，Us 事實上是一直在 Document Server 交談，跟本不知道 Ss 的狀態，若此時 Ss 其實已經走掉了，Us 也無從得知。

XAML-V 以<proxy-request>語法來實現此一構想。每一份 XAML-V 的對話文件都會含有一個 Form 元素，Form 元素包含零到多個 Field 元素，Field 代表預期使用者回應的資訊。XAML-V 解譯器在取得使用者的回應，得到所有 Field 的

答案後，會將這些資訊以 HTTP Request 的型式加以封裝，向 Document Server 發出 GET 或 POST 要求。HTTP Request 中包含 Field 的名稱與使用者所填入的資訊，以多組 Key-Value 的對應(Map)型式存在。

在新的對話模型中，Us 在取得使用者回應後，必須先將 HTTP Request 的資訊傳送給 Ss，再委由 Ss 來代為執行 HTTP GET 或 POST。因此我們定義了 Proxy Request 的訊息格式來封裝 Us 的 HTTP Request。

```
<xaml-v>
    <protocol>
        <proxy-request>
            <method>GET</method>
            <url>helloFormResponse.jsp</url>
            <parameters>
                <param key="helpType" value="no thanks"/>
            </parameters>
        </proxy-request>
    </protocol>
</xaml-v>
```

圖 4.16　Us 傳送一個 proxy request 訊息給 Ss

一個 HTTP Request 會包含三個部份的資訊:request method、request url 與 request parameters。我們在 XAML-V 中定義了一個 proxy-request 元素來表達這些資訊。圖 4.16 是一個 proxy-request 訊息的範例，其中 request method 是 GET，request url 是 helpFormResponse.jsp，這是一個 context relative url，會由 Ss 根據前一份文件的 url 所在位置轉換成絕對 url。最後一部份 request parameters 則是由一群 key-value 所組成的參數對，用來表達使用者在前一段對話所做的回應。

綜合本章論述，我們認為 XAML-V 具有下列優點與特色：

1.　XAML-V 以動畫腳本語言 XAML 為主體並加以擴充，得以保留 XAML 原

有強大的動畫功能，符合 3D 多人虛擬環境的特色。

2. XAML-V 使用 Server-side Script，不但簡化了對話腳本語言的複雜度，並得以和現有主流伺服端動態腳本技術如 JSP、ASP 等相容。

3. *大部份的採用 VoiceXML 做對話腳本語言之相關研究均受限於 VoiceXML 的 Dialog Model，只能在單人環境下運作。*XAML-V 針對多人環境的需求，對 VoiceXML 的 Dialog Model 加以擴充，*可在多人環境中運作。*