第四章　輔助設計系統建構

本章將承續第三章分析後獲致之結論，將重點置於擴充架構之設計來進行系統建構。首先選擇一個合適的輔助設計操作環境及程式開發工具，進而使用開發工具撰寫前述性質分析所得之結論，以“風”爲例，編寫環境元件程式碼；再來運用之前討論所得的知識擴充架構，編撰可於實質操作環境中操作的環境資料結構程式碼。最後將虛擬的建築元件及“風”元件資料輸入完成之系統，觀察知識擴充架構的運作結果，對於環境設計知識的運用是否有實質上的作用，以作爲具備環境與人爲元件互動的輔助設計系統中，知識呈現方式的參考資料。

4-1 輔助設計環境及程式開發工具之選擇

輔助設計研究環境選擇

本論文之研究重點並非建立一個完整的輔助設計系統，而是其中設計知識呈現的架構部分，所以首先應選擇一個穩定而成熟的輔助設計系統作爲系統開發之研究環境，如此才能減少不相關的程式撰寫，將重心置於研究主題。而此輔助設計系統應具備可供程式開發的介面，以利於開發架構程式與系統的整合。此外還能夠提供一個簡易的操作環境及結果呈現，以作爲實質運作的檢驗。至於系統相關資料之多寡與取得的困難度也是考慮的重要因素之一。所以依據上面所列之考慮因素，以個人觀點，就現有之輔助設計系統作了簡單的評估：

評估項目 / 系統名稱	穩定性	程式開發介面	資料來源	操作簡易	結果呈現
AutoCAD	佳	多	多	可	佳
MicroStation	佳	少	少	可	佳
Form Z	佳	少	少	可	佳
ArchiCAD	可	少	少	佳	佳

表 4-1 輔助設計系統選擇評估表

由於本研究方式在於程式開發，因而穩定性、程式開發介面及資料來源爲選擇開發環境系統之重點。經由上表之評估後，因開發介面多且具備充足的資料來源，所以選擇了 AutoCAD 作爲開發環境。

研究程式開發工具選擇

選擇了 AutoCAD 作爲開發環境系統，緊接著就是選擇此環境所提供的何種程式介面作爲開發工具。選擇開發工具時，應注意到的是所要建立的元件本質該如何以程式表達。就人造物與環境元件而言，是以設計原型爲知識呈現基本概念，而設計原型所引用的基模概念在程式語言的觀念上就是一種類別概念，而人造及環境元件類似類別衍生的物件，元件的性質就可以物件中的變數及函式表達，所以基本選擇要件是最好具備物件導向設計的程式開發工具，其次則是考慮系統開發之難易度、執行速度與系統整合度。以下就 AutoCAD 所提供的各種開發工具，列出一個評估表以作爲選擇參考的依據。

評估項目 / 系統名稱	物件設計能力	開發難易度	執行速度	系統整合度
AutoLISP	無	易	慢	佳
Visual LISP	無	易	可	佳
AutoCAD VBA	需透過 Active X 介面	可	可	佳
ObjectARX	有	不易	快	佳

表 4-2 輔助設計系統開發工具選擇評估表

由於本研究開發系統最好需具備物件設計能力，其中 ObjectARX 雖然開發較爲不易，但僅有其具備完整的物件設計能力，所以還是選擇以 ObjectARX 作爲系統開發工具。

4-2 輔助系統之階層式元件設計

環境元件設計

在 3-5 節中已對自然元件的原型作了很明確的定義，所以現在就依據這個定義進行實際的元件程式撰寫，但由於研究重點是在輔助設計系統的擴充架構上，

而不是在於對人類設計者的資訊提供，所以在元件型式部分就不再作深入的表達，而著重於將元件以機能及行爲來表達本身的意義。

而單就“風”這個環境元件的物件類別定義，就可以運用程式表達如下：

```
class Wind : public AcDbObject
//
// This class demonstrates wind objects.
//
// To use it in arch environment, this class will affect other architecture component.
//
{
    public:
        ACRX_DECLARE_MEMBERS(Wind);
        Wind(): dSpeed(0.0), iDirection(0) {};
        Wind(const double& dSpd, const int& iDir): dSpeed(dSpd), iDirection(iDir) {};

       Acad::ErrorStatus        getData     (double&, int&);
        Acad::ErrorStatus        setData     (double, int);

       virtual Acad::ErrorStatus dwgInFields (AcDbDwgFiler*);
        virtual Acad::ErrorStatus dwgOutFields(AcDbDwgFiler*) const;
//      virtual Acad::ErrorStatus dxfInFields (AcDbDxfFiler*);
//      virtual Acad::ErrorStatus dxfOutFields(AcDbDxfFiler*) const;
    private:
        double dSpeed;
        int iDirection;
};
```

表 4-3 自然環境“風”物件類別基本定義

單元階層式架構設計

在目前的 CAD 系統中主要仍以幾何圖形元件爲主，AutoCAD 也不例外。而爲了將階層式架構整合於 AutoCAD 中，以利於在真實輔助設計系統中觀察階層架構的實質變化，因而配合 AutoCAD 的資料架構作調整，將階層式架構於非圖形元件的資料結構區中，以實質意義作關係架構之建立；而建築型式元件呈現則在 AutoCAD 本身的圖形資料區中，以幾何圖形元件作型式表達；然後再將階層架構單元與幾何圖形元件建立連結。所以就可以將基本單元的階層式資料架構設

計說明如下。

首先在非圖形資料區部分，由於環境元件設計部分暫不考慮型式表現的方式，所以將環境元件（如“風”物件）運用串列索引資料結構 Dictionary，建立索引連結，而將單元架構設計如下。

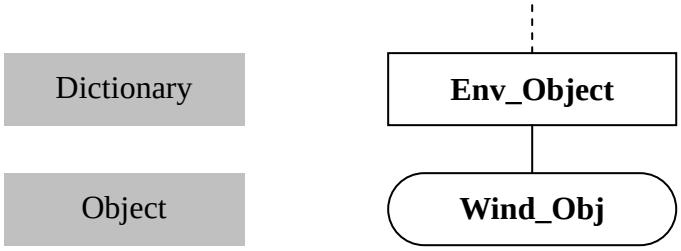

圖 4-1 環境元件類階層式單元架構（Dictionary 爲索引資料結構型式）

而在非圖形資料區的建築元件設計部分，則要考慮關於形式表現的方式，所以必須考慮對於幾何圖形的連結關係，因而除了運用串列索引資料結構 Dictionary 連結建築元件外，另外則將建築元件本身設計爲一個特殊的資料結構 Entry，內含型式表現的幾何元件索引（Archobj ID），用以建立階層元件與幾何元件間的單向連結關係。除此之外，還需表達最重要階層式架構行爲，因而於 Entry 結構下再聯結了延伸式串列索引資料結構 Extended Dictionary，所以單元架構設計方式與環境元件也就有所不同：

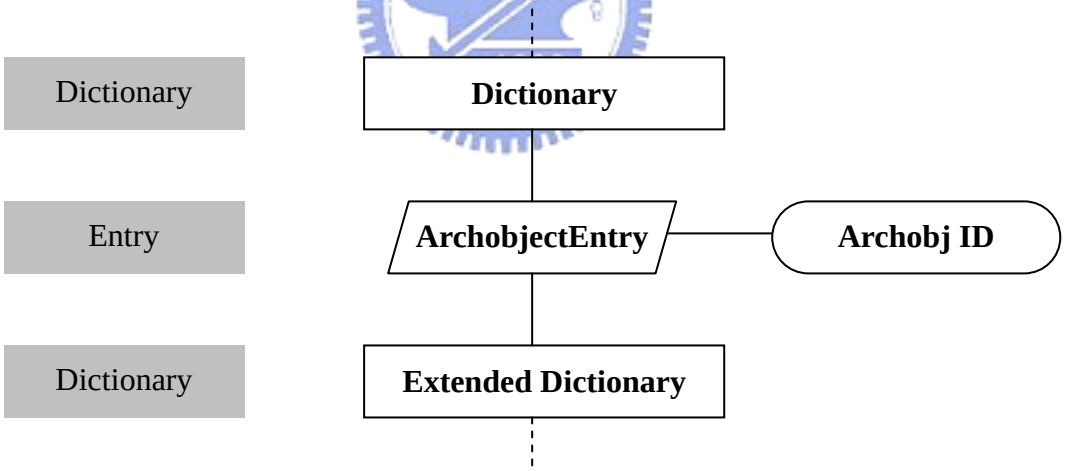

圖 4-2 建築元件類階層式單元架構（Entry 爲圖形連結索引資料結構型式）

依據這個架構，就可將 Entry 類別以程式碼表達如表 4-2：

非圖形資料區部分之元件設計完成後，接下來就針對圖形資料區的幾何元件進行設計。處理建築圖形元件設計時，除了基本幾何資料外，也需對幾何圖形作建築元件的人爲定義，如此一般性的幾何元件才能具有建築元件的實際性質。所以設計了建築元件的資料結構 ArchobjDetails，其中包含了基本建築元件名稱，此外如前述的 Entry 資料結構一樣，也內含階層架構的建築元件索引(Archobj Entry

ID）以建立單向聯結至階層架構中的建築元件 Entry。至於 ArchobjDetails 資料結構與圖形元件間則運用延伸式串列索引資料結構 Extended Dictionary 加以連結。其設計單元架構如圖 4-3：

```
//Entry OBJECT is defined in the ArchObject project
//export the class depending on the
//_ARCHOBJECT symbol
class
#ifdef _ARCHOBJECT
_declspec(dllexport)
#endif
ArchobjEntry : public AcDbObject{
protected:
    AcDbObjectId m_idArchobj;
    static long version;
public:
    ACRX_DECLARE_MEMBERS(ArchobjEntry);
    const AcDbObjectId ArchobjId()const;
    Acad::ErrorStatus setArchobjId(const AcDbObjectId& idArchobj);

    //AcDbObject overrides
    virtual Acad::ErrorStatus dwgInFields (AcDbDwgFiler* filer);
    virtual Acad::ErrorStatus dwgOutFields(AcDbDwgFiler* filer) const;

#ifdef MEM_DEBUG
#undef new
#undef delete
#endif
    void *operator new[](unsigned nSize) { return 0;}
    void operator delete[](void *p) {};
    void *operator new[](unsigned nSize, const char *file, int line) { return 0;}
#ifdef MEM_DEBUG
#define new DEBUG_NEW
#define delete DEBUG_DELETE
#endif
};
```

表 4-4 階層式建築物件類別基本定義

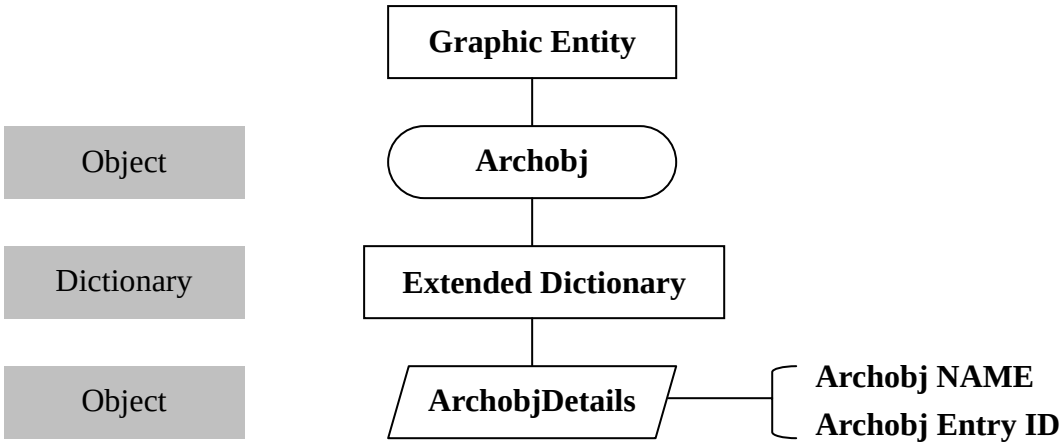

圖 4-3 建築幾何圖形元件單元架構

依據上面架構，就可將 ArchobjDetails 類別以程式碼表達如表 4-3：

```
class
#ifdef _ARCHOBJECT
_declspec(dllexport)
#endif
ArchobjDetails : public AcDbObject{
protected:
    char* m_strArchobjName;
    AcDbObjectId m_idArchobjEntry;
    static long version;
    void cleanUp();
public:
    ACRX_DECLARE_MEMBERS(ArchobjDetails);
    ArchobjDetails();

    virtual ~ArchobjDetails();

    //ArchobjDetails protocol
    const char* ArchobjName() const;
    Acad::ErrorStatus setArchobjName(const char* strArchobjName);
    const AcDbObjectId ArchobjEntryId()const;
    Acad::ErrorStatus setArchobjEntryId(const AcDbObjectId& idArchobj);
```

表 4-5 建築幾何圖形元件單元類別基本定義

```
    //AcDbObject overrides
    virtual Acad::ErrorStatus dwgInFields (AcDbDwgFiler* filer);
    virtual Acad::ErrorStatus dwgOutFields(AcDbDwgFiler* filer) const;

#ifdef MEM_DEBUG
#undef new
#undef delete
#endif
    void *operator new[](unsigned nSize) { return 0;}
    void operator delete[](void *p) {};
    void *operator new[](unsigned nSize, const char *file, int line) { return 0;}
#ifdef MEM_DEBUG
#define new DEBUG_NEW
#define delete DEBUG_DELETE
#endif
};
```

續表 4-5 建築幾何圖形元件單元類別基本定義

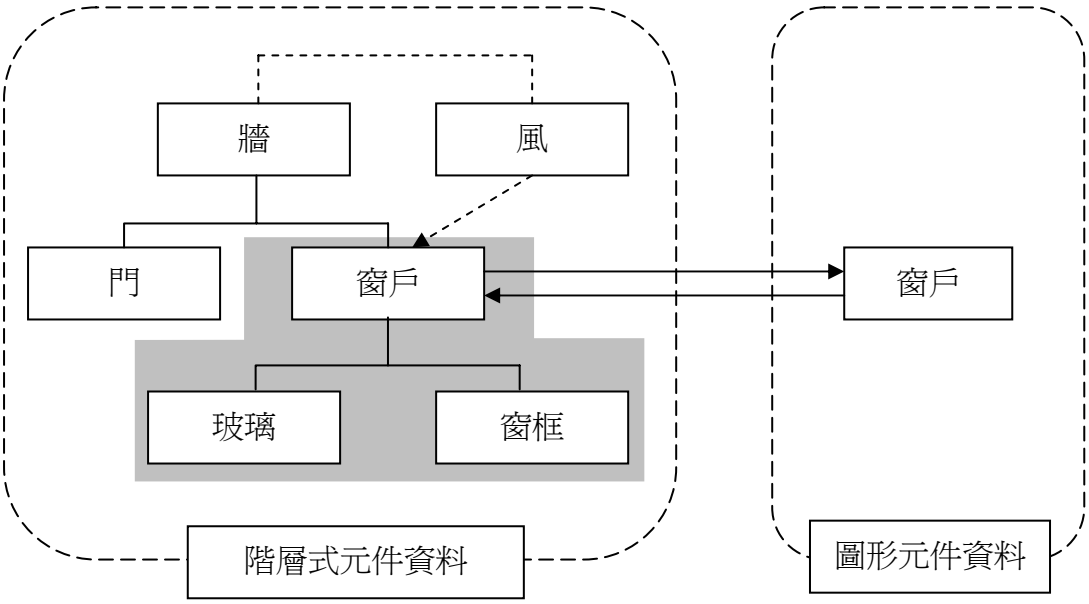

圖 4-4 幾何圖形元件與階層元件之互動關係