Automation and IDispatch

Jim Fawcett

CSE775 – Distributed Objects

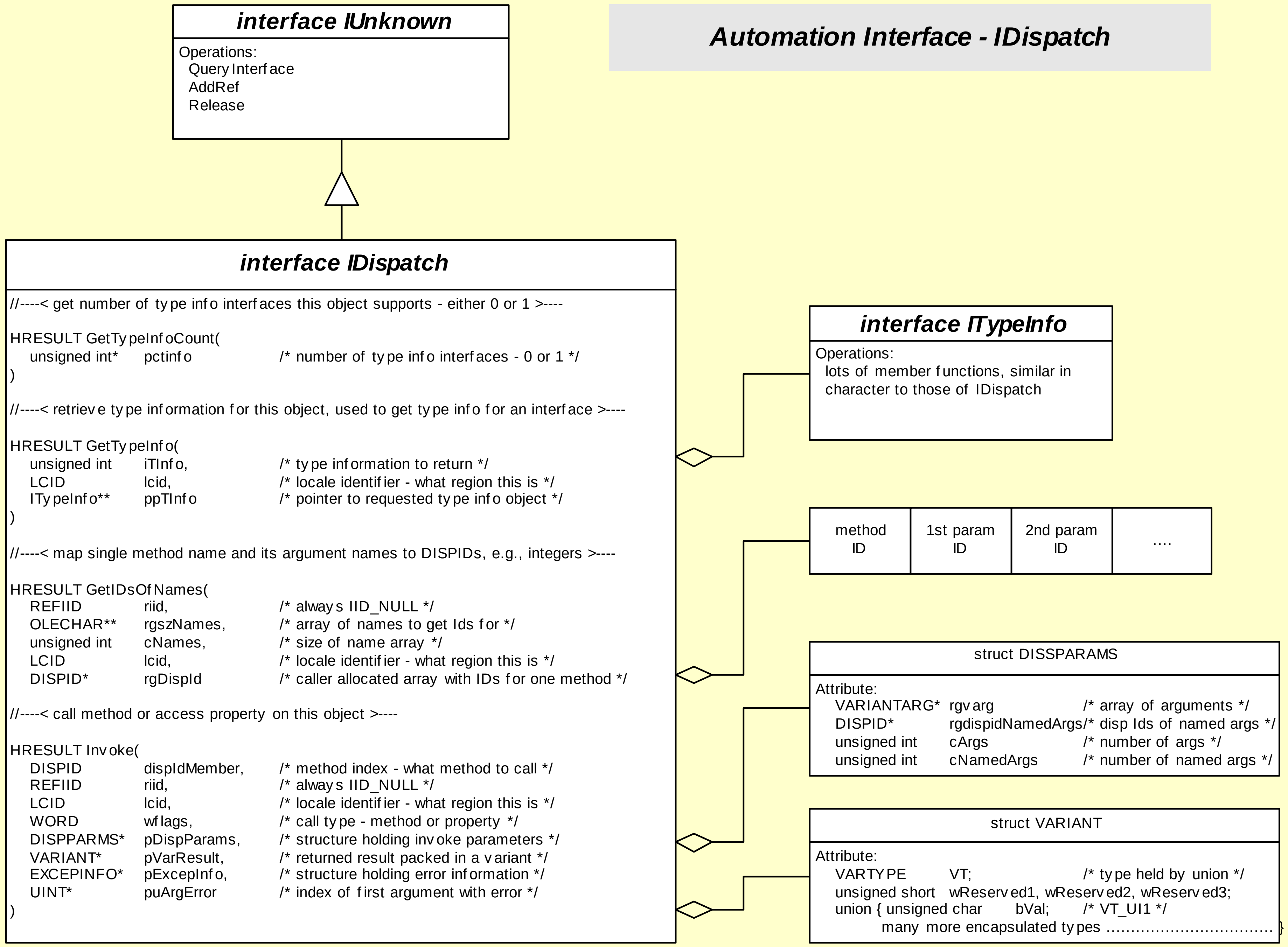
Spring 2004

Automation

- An automation server is a COM object that exposes methods and properties through the IDispatch interface.

- IDispatch is a “generic” interface
  - Used by scripting languages that have no mechanism for compiling information about an interface.
  - Its invoke() method is passed an index into a table of methods.
  - It uses a DISSPARMS structure to pass parameter values to the interface.
  - It uses a VARIANT tagged union to return method results.

Automation Interface - IDispatch



interface IUnknown

```
Operations:
  QueryInterface
  AddRef
  Release
```

interface IDispatch

```
//----< get number of type info interfaces this object supports - either 0 or 1 >----

HRESULT GetTypeInfoCount(
    unsigned int*     pctinfo                /* number of type info interfaces - 0 or 1 */
)

//----< retrieve type information for this object, used to get type info for an interface >----

HRESULT GetTypeInfo(
    unsigned int      iTInfo,                /* type information to return */
    LCID              lcid,                  /* locale identifier - what region this is */
    ITypeInfo**       ppTInfo                /* pointer to requested type info object */
)

//----< map single method name and its argument names to DISPIDs, e.g., integers >----

HRESULT GetIDsOfNames(
    REFIID            riid,                  /* always IID_NULL */
    OLECHAR**         rgszNames,             /* array of names to get Ids for */
    unsigned int      cNames,                /* size of name array */
    LCID              lcid,                  /* locale identifier - what region this is */
    DISPID*           rgDispId               /* caller allocated array with IDs for one method */

//----< call method or access property on this object >----

HRESULT Invoke(
    DISPID            dispIdMember,          /* method index - what method to call */
    REFIID            riid,                  /* always IID_NULL */
    LCID              lcid,                  /* locale identifier - what region this is */
    WORD              wflags,                /* call type - method or property */
    DISPPARMS*        pDispParams,           /* structure holding invoke parameters */
    VARIANT*          pVarResult,            /* returned result packed in a variant */
    EXCEPINFO*        pExcepInfo,            /* structure holding error information */
    UINT*             puArgError             /* index of first argument with error */
)
```

interface ITypeInfo

```
Operations:
  lots of member functions, similar in
  character to those of IDispatch
```

method ID	1st param ID	2nd param ID	....

struct DISSPARAMS

```
Attribute:
   VARIANTARG*  rgvarg               /* array of arguments */
   DISPID*      rgdispidNamedArgs/* disp Ids of named args */
   unsigned int cArgs                /* number of args */
   unsigned int cNamedArgs           /* number of named args */
```

struct VARIANT

```
Attribute:
   VARTYPE        VT;                /* type held by union */
   unsigned short wReserved1, wReserved2, wReserved3;
   union { unsigned char     bVal;     /* VT_UI1 */
           many more encapsulated types ................................. }
```

The Variant Data Type

- A Variant variable is capable of storing all system-defined types of data. You don't have to convert between these types of data if you assign them to a Variant variable; Visual Basic 6.0 automatically performs any necessary conversion.

- For example:

```
Dim SomeValue
    ' Variant by default.
SomeValue = "17"
    ' SomeValue contains "17‟ (a two-character string).
SomeValue = SomeValue - 15
    ' SomeValue now contains the numeric value 2.
SomeValue = "U" & SomeValue
    ' SomeValue now contains ' "U2" (two- character string).
```