Costruzione di Interfacce
Lezione 26
Xml for dummies

cignoni@iei.pi.cnr.it
http://vcg.iei.pi.cnr.it/~cignoni

XML

- Cos'è XML?
- acronimo di eXtensible Markup Language
- è un linguaggio estensibile realizzato per poter definire ed utilizzare in modo semplice documenti strutturati
- Nato e studiato per il Web,
- Molto utile al difuori del contesto web per definire generici formati di interscambio di dati.
- XML è molto di più di quello che vedrete qui…

Esempio xml

```
<?xml version="1.0" encoding="utf-8"?>
<business-card>
  <persona>
    <titolo> Sig. </titolo>
    <nome> Mario </nome>
    <cognome> Rossi </cognome>
  </persona>
  <indirizzo>
    <strada> Via Verdi </strada>
    <numero-civico> 12 </numero-civico>
    <cap> 56100 </cap>
    <citta> Pisa </citta>
  </indirizzo>
</business-card>
```


Xml

- Basato sull'uso di markup per delimitare porzioni del documento
- Un mark-up è quello compreso tra <>
- Detto anche tag, o etichetta
- I tag devono essere annidati
- XML non ha tag predefiniti
  - è estensibile
  - consente di definire nuovi linguaggi
  - è un metalinguaggio
- Uno degli obiettivi di progettazione di XML:
  - deve essere in un formato leggibile dall'uomo
  - è ascii.

Xml

I documenti xml sono strutturati in tre parti, (le prime due sono opzionali)

1. *XML processing instruction:* identifica la versione di XML usata e come è codificata e se riferisce altri file:
   `<?xml version="1.0" encoding="UCS2" standalone="yes"?>`
2. *document type declaration (DTD):* che o contiene al suo interno una dichiarazione formale di quali tag (e come) il doc usa (*internal subset*), o riferisce un file che contiene tali dichiarazioni: (*external subset*),
   `<!DOCTYPE memo SYSTEM "http://www.myco.com//memo.dtd">`
3. Il doc xml vero e proprio il cui elemento radice corrisponde con il tipo specificato nel dtd. Tutti gli altri tag sono nestati in questo.

Xml dettagli

Sintassi
- È case sensitive

Elemento: ciò che è racchiuso da una coppia <tag> … </tag>
- Un tag può avere *attributi*
  <tag att0="str1" att0="str1" > </tag>
- Valida anche la sintassi
- <tag attr="asdfasd" />

Commenti
- <!-- questo e' un commento. //-->

XML

- Perchè usiamo xml?
- Perchè aiuta molto nella definizione del formato di file della nostra applicazione
  - No problemi di parsing.
  - Estendibilità innata (è facile fare backward compatibility)
    - Facile gestire set non ordinati
    - Facile gestire assenza di elementi
  - Fa sempre comodo conoscere una tecnologia in più

XML e linguaggi oo

- Ad ogni classe corrisponde un elemento
- I membri di una classe sono naturalmente annidati.
- Corrispondenza naturale tra strutture dati e formato xml

Esempio

```
<business-card>
  <persona>
    <titolo> Sig. </titolo>
    <nome> Mario </nome>
    <cognome> Rossi
    </cognome>
  </persona>
  <indirizzo>
    <strada> Via Verdi
    </strada>
    <numero-civico> 12
    </numero-civico>
    <cap> 56100 </cap>
    <citta> Pisa </citta>
  </indirizzo>
</business-card>
```

```
class BusinnesCard
{ public:
    Persona m_pers;
    Indirizzo m_ind;
};
class Persona
{public:
    string m_titolo;
    string m_nome;
    string m_cognome;
};
class Indirizzo
{public:
    string m_strada;
    string m_numero
    int m_cap;
string m_citta;
}
```


Xml in pratica

- Generare xml è assai facile
- La via del praticone:
  - printf("<tag> %i </tag>",val);
  - Facile ma possibilità di fare errori di sintassi xml…
- Appoggiandosi implicitamente al Linguaggio/environment
  - Se si passa a Managed c++, si può dichiarare una classe con attributo
  - [System::Serializable]
  - E far fare la serializzazione al framework

<managed c++>

- Introdotto in .Net
- Via di mezzo tra c++ e c#
- Definibile approssimativamente come un c++ con un garbage collector stile java applicabile solo a determinati oggetti e alcune cose compilate jit da MSIL (microsoft intermediate language);
- Basta:
  - Aggiungere davanti ad una classe __gc
  - Compilare con opzione /clr (compile as managed)
  - Aggiungere #using<mscorlib.dll>

<managed c++>

- Esempio minimo minimo di serializzazione automatica in managed c++

```
#using <mscorlib.dll>
#using <System.xml.dll>
[System::Serializable]
public __gc class test {public:
   double ssadf;
};
…
test* po = new test();
po->ssadf= 1234.0;
System::Xml::Serialization::XmlSerializer* ser =
   new System::Xml::Serialization::XmlSerializer(po->GetType());
System::IO::TextWriter* writer = new
   System::IO::StreamWriter("xx.xml");
ser->Serialize(writer,po);
writer->Close();
```

</managed c++>

- L'esempio precedente genera (più o meno):

```
<?xml version="1.0" encoding="utf-8"?>
<test >
  <ssadf>1234</ssadf>
</test>
```

- Bello, ma con un sacco di limitazioni su quali classi posso fare __gc e con varie penalty di performance dovute all'uso di classi compilate in msil…

Modello DOM

- I parser xml appartengono a due grosse categorie
- DOM: Document Object Model
  - Leggono tutto un file xml e restituiscono un oggetto c++ (o in qualunque altro linguaggio) che memorizza tutto l'albero con tutti gli elementi e gli attributi.
  - L'applicazione poi naviga l'albero con calma.
- SAX: Simple Api for Xml
  - Generano eventi durante la lettura che devono essere gestiti dall'applicazione

Yet another xml parser

- Di parser xml ne esistono molti (sia microsoft, che legati al linguaggio)
- in vcg/xml/xml.h trovate una classe xmldoc che implementa un parser dom minimo,
- nel senso di numero di linee di codice.
- Basata principalmente sulle stl
- Non efficientissima, ma leggibile (~300 righe di codice tra .h, .y e .l)

Vcg/Xml

- Due classi:
  - XmlDoc
    - Rappresenta tutto un documento xml dopo che è stato parsato da un file
    - XmlDoc xd; xd.read("myfile.xml");
    - La radice dell'albero xml è in Xd.start;
  - Xml nodo dell'albero;
  - Ogni nodo contiene:
    - string id; // il tag
    - string content;
    - map<string,string> attr;
    - vector<Xml> children;

Vcg/xml

- Dopo che si è parsato un file xml, occorre fare a mano la creazione delle classi corrispondenti ai vari nodi
- Molto facile se si fa tutto con la stessa interfaccia.
- Per ogni classe che si vuole rendere persistente si aggiunge due funzioni
  - XMLWrite(FILE *fp)
  - XMLRead(Xml& xml);
- E si usa tutto ricorsivamente

Scene Graph

- Il problema principale è che il mio documento/scena non è affatto strutturato.
- Scriviamoci il nostro semplicissimo scene graph ad hoc
- Cosa ci deve stare dentro?
  - Mesh
  - trasformazioni
  - Animazioni

CSG

- La scena la assumo strutturata come un albero
- Assumo che ogni nodo sia riferito solo una volta.
- Due classi principali
  - CSG: contiene l'intero scene graph (la radice)
  - CSGNode: generico nodo dello scene graph, lo faccio come classe astratta;

CSGNode

```
class CSG
{public:
   CSG(void);
   ~CSG(void);
   CSGGroup root;
};
class CSGNode
{public:
   virtual ~CSGNode(void){};
   virtual void glDraw(const float DocTime)=0;
};
```

- Notare che
  - La draw prende in ingresso un tempo che verrà usato o meno
  - La draw è una funzione pure virtual, quindi la classe CSGNode non è istanziabile
  - Tutto quello che mi serve lo derivo da CSGNode, specializzando distruttore e glDraw.

CSGGroup

- Nodo principale per rappresentare un gruppo di nodi
- È quello che definisce lo scope delle trasformazioni di modellazione
- Si assume che i figli possano cambiare la modelview modificando lo stato per i successivi fratelli
- L'ordine dei figli è significativo
- È responsabile della disallocazione dei nodi.

CSGGroup

```
class CSGGroup :public CSGNode
{  public:
  virtual ~CSGGroup();
  typedef list<CSGNode *>::iterator iterator;
  list<CSGNode *> Sons;
  virtual void glDraw(const float DocTime);
};

CSGGroup::~CSGGroup()  //distruttore: disalloca tutti i figli
{
  for(iterator i=Sons.begin();i!=Sons.end();++i)
        delete (*i);
}

void CSGGroup::glDraw(const float DocTime)
{
  glPushMatrix();
  for(iterator i=Sons.begin();i!=Sons.end();++i)
    (*i)->glDraw(DocTime);
  glPopMatrix();
}
```


CSGTransformation

Classe che incapsula una generica, statica trasformazione (o composizione di varie trasformazioni)

```
class CSGTransformation :public CSGNode
{  public:
  Matrix44f m;
  virtual void glDraw(const float DocTime);
};

void CSGTransformation::glDraw(const float DocTime)
{ glMultMatrix(m); };
```


CSGAnimRotation

- Classe che incapsula una rotazione animata, notare che il tempo da visualizzare arriva da fuori.

```
class CSGAnimRotation :public CSGNode
{ public:
  Point3f axis;
  float AngularSpeedDPS; //Degree Per Sec;
  float StartAngleDeg;
  virtual void glDraw(const float DocTime);
};

void CSGAnimRotation ::glDraw(const float DocTime)
{
   float CurAngleDeg = StartAngleDeg + DocTime*AngularSpeedDPS ;
   glRotatef(CurAngleDeg , axis[0],axis[1],axis[2]);
};
```

CSGAnimZPrecession

- Moto di precessione animato sull'asse Z

```
class CSGAnimZPrecession :public CSGNode
{public:
  CSGAnimZPrecession();
  virtual ~CSGAnimZPrecession(){};
  float DeclinationDeg;
  float AngularSpeedDPS; //Degree Per Sec;
  float StartAngleDeg;
  virtual void glDraw(const float DocTime);
};
CSGAnimZPrecession::CSGAnimZPrecession()
{  StartAngleDeg=0;AngularSpeedDPS=10;DeclinationDeg=30;}
void CSGAnimZPrecession::glDraw(const float DocTime)
{ // precessione: una rotazione il cui asse ruota intorno all'asse z
   float CurAngleRad=ToRad(StartAngleDeg + DocTime*AngularSpeedDPS);
   glRotated( DeclinationDeg,cos(CurAngleRad),sin(CurAngleRad),0);
};
```


Modifichiamo MoebiusStrip

- Deriviamola da CSGNode
  - In questo modo diventa un oggetto che può far parte dello scene graph.
- Adattiamo la funzione draw
- Aggiungiamo il distruttore virtuale (che non fa nulla)

Usiamo lo SceneGraph

- Nel doc aggiungiamo un membro:
- `CSG Scene;`
- OnNewDocument: aggiungiamo la costruzione dello scene graph
- `Scene.root.Sons.push_back(new CSGAnimZPrecession());`
- `MoebiusStrip *ms=new MoebiusStrip();`
- `ms->Generate();`
- `Scene.root.Sons.push_back(ms);\`

- Notare come si crei apposta un oggetto MoebiusStrip con la new anziché passargli l'indirizzo del membro del doc. Questo per evitare che poi il distruttore del gruppo faccia pasticci cercando di disallocare qualcosa non allocato con la new

Usiamo lo SceneGraph

- Nella OnDraw della classe GLView buttiamo via tutto quello dopo la trasf della trackball e aggiungiamo:

- `pd->Scene.root.glDraw(pd->ElapsedSec);`

- Si dovrebbe fare la stessa cosa con tutto il resto (omini, pallina ecc)

Salviamo in XML

- Seguiamo la via del praticone:
- Aggiungiamo al nodo base

```
virtual void XMLWrite(FILE *fp)=0;
```

- Che ci obbliga ad implementare tale metodo in TUTTI gli altri nodi da esso derivati

```
void CSGGroup::XMLWrite(FILE *fp)
{
   fprintf(fp,"<CSGGroup >\n");
   for(iterator i=Sons.begin();i!=Sons.end();++i)
        (*i)->XMLWrite(fp);
   fprintf(fp,"</CSGGroup>\n");
}
```


XMLWrite

```
void CSGAnimZPrecession::XMLWrite(FILE *fp)
{
  fprintf(fp,"<CSGAnimZPrecession\n");
  fprintf(fp,"  DeclinationDeg = \"%f\"\n", DeclinationDeg);
  fprintf(fp,"  AngularSpeedDPS = \"%f\"\n", AngularSpeedDPS );
  fprintf(fp,"  StartAngleDeg = \"%f\"\n", StartAngleDeg);
  fprintf(fp,"/>\n");
}
void MoebiusStrip::XMLWrite(FILE *fp)
{
   fprintf(fp,"<MoebiusStrip\n");
   fprintf(fp,"  BlockNum = \"%i\"\n",BlockNum);
   fprintf(fp,"/>\n");
}
```

- Il parsing alla prossima puntata!