*UNIVERSIDADE NOVA DE LISBOA*
FCT – Departamento de Engenharia Electrotécnica
**Secção de Robótica e Manufactura Integrada**

# Integração de Sistemas

2007 / 2008

## Trabalho 2

## Utilização de XML

3 aulas x 3 horas
Data de entrega – 11 de Abril de 2008

## 1. INTRODUÇÃO

Hoje em dia existe um amplo conjunto de tecnologias que permitem de forma, mais ou menos complexa, a troca de informação entre e intra sistemas. A crescente utilização de redes de equipamentos, onde o poder computacional se encontra distribuído por diversas máquinas, tem permitido maximizar a troca de informação e contribuído para a constituição de novas formas de negócio, designadamente a oferta de serviços electrónicos (e-activities: e-business, e-maintenance, e-monitoring, etc). No entanto, a heterogeneidade dos sistemas levanta problemas significativos de integração (como integrar sistemas que usam diferentes: tecnologias e formatos de dados? ). A uniformização da informação desempenha neste contexto um papel significativo. O XML surgiu como um mecanismo de efectuar essa uniformização. Tratando-se de uma linguagem textual, genérica e baseada em marcas (tags), definidas pelo utilizador, o XML pode ser facilmente interpretado por todo um conjunto de ferremantas disponíveis para diversas plataformas e linguagens (Windows, Linux, UNIX, C#, C, C++, JAVA, etc). Com a vantagem que para ficheiros de dimensão média é legível por humanos.
No presente trabalho vamos abordar a utilização de XML para armazenamento de dados (pequena base de dados) e comunicação entre aplicações (troca de mensagens formatadas em XML). Um aspecto importante a considerar é o formato do ficheiro XML. Devido à total liberdade da linguagem os ficheiros podem conter literalmente "qualquer coisa" (desde que as marcas estejam correctamente formatadas). Para melhorar as aplicações recorre-se-á a XML schemas que são ficheiros XML que validam a forma e conteúdo dos ficheiros e mensagens.

## 2. APRESENTAÇÃO DO PROBLEMA

### 2.1 NovaWars (Mecânica do Jogo)

O presente trabalho consiste no desenvolvimento de um cliente para o jogo "NovaWars". O cliente será re-utilizado no trabalho seguinte da cadeira pelo que deve haver cuidado no desenvolvimento do mesmo.
O jogo "NovaWars" é um jogo, baseado em ASP.Net Web Services, de batalhas épicas em que cada jogador (cliente) combate com os demais utilizando armas diferentes pela conquista do título "NovaKing".

Cada jogador possui uma palavra passe e um id que o autenticam, uma pontuação de vida e dinheiro e um conjunto de armas .

Será distribuída uma biblioteca, que trata dos detalhes de interacção com o servidor, e disponibiliza os seguintes métodos:

```
public string RegisterPlayer(string playerAlias, string password)
```

Deve ser a primeira operação invocada. Permite registar a personagem com o nome `playerAlias` e associa-lhe uma palavra passe `password` para autenticação no servidor em futuras utilizações. O servidor devolve uma mensagem com o estado da operação. (ver abaixo a secção que detalha as mensagens XML recebida e enviadas do e para o servidor).

```
public string GetPlayerList(string xmlFormatedMessage, string password)
```

Após o envio de um mensagem correctamente formatada `xmlFormatedMessage` e uma palavra passe `password` válida servidor o devolve uma mensagem com a listagem de todos os jogadores (ver abaixo a secção que detalha as mensagens XML recebida e enviadas do e para o servidor).

```
public string Fight(string xmlFormatedMessage, string password)
```

Após o envio de um mensagem correctamente formatada `xmlFormatedMessage` e uma palavra passe `password` válida o servidor devolve uma mensagem com o resultado de uma batalha (ver abaixo a secção que detalha as mensagens XML recebida e enviadas do e para o servidor).

```
public string GetStatus(string xmlFormatedMessage, string password)
```

Após o envio de um mensagem correctamente formatada `xmlFormatedMessage` e uma palavra passe `password` válida o servidor devolve uma mensagem com o estado de um jogador (ver abaixo a secção que detalha as mensagens XML recebida e enviadas do e para o servidor).

```
public string GetCatalogList(string xmlFormatedMessage, string password)
```

Após o envio de um mensagem correctamente formatada `xmlFormatedMessage` e uma palavra passe `password` válida o servidor devolve o catálogo de armas e artefactos que o jogador pode comprar (ver abaixo a secção que detalha as mensagens XML recebida e enviadas do e para o servidor).

```
public string Buy(string xmlFormatedMessage, string password)
```

Após o envio de um mensagem correctamente formatada `xmlFormatedMessage` e uma palavra passe `password` válida o servidor devolve a arma comprada pelo jogador(ver abaixo a secção que detalha as mensagens XML recebida e enviadas do e para o servidor).

Assim compete a cada jogador fazer a gestão dos combates e armas. A cada arma está associada uma energia. Cada vez que essa arma entra em combate a sua energia diminui até que a arma deixa de ser eficaz e é destruída pelo servidor. Quando um jogador ataca dependendo do sucesso do ataque a sua energia e dinheiro podem aumentar ou diminuir. O jogador vencedor será aquele que ao longo do jogo conseguir atingir a maior pontuação ponderada de vida e dinheiro.

Neste contexto utilizar-se-á XML a para resolver vários problemas:

1. Manter uma base de dados actualizada com o registo das batalhas.
2. Manter uma base de dados com o armamento actual.
3. Trocar mensagens com o servidor.
4. Validar os ficheiros da base de dados e de mensagem (recurso a XML schemas).

### 2.2 NovaWars (Comunicação com o Servidor)

A comunicação com o jogo faz-se por intermédio da biblioteca anteriormente descrita. Exceptuando o método de registo do jogador, todos os demais utilizam a variável `string xmlFormatedMessage` para comunicar com o servidor. Esta variável tem um conteúdo XML serializado na forma de uma cadeia de caracteres `string`.

Do ponto de vista da descrição XML a mensagem deve obedecer ao seguinte Schema:

```
<xs:element name ="Message">
    <xs:complexType>
      <xs:sequence minOccurs="1" maxOccurs="1">
        <xs:element name ="Sender" type="tMsgEntity"/>
        <xs:element name="Receiver" type="tMsgEntity"/>
        <xs:element name="Content" type="tMsgContent"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

Explicação: cada mensagem possui um emissor `Sender` (que contém o identificador `playerAlias` do jogador), um receptor `Receiver` (que identifica o destinatário neste caso um nome qualquer associado ao servidor) e um conteúdo `Content`.

```
<xs:complexType name="tMsgEntity">
    <xs:sequence>
      <xs:element name="Name" type ="xs:string"/>
    </xs:sequence>
  </xs:complexType>
```

Explicação: o emissor `Sender` e o receptor `Receiver` são nome do tipo cadeia de caracteres `xs:string`.

```
<xs:complexType name="tMsgContent">
    <xs:sequence minOccurs="1" maxOccurs="1">
      <xs:element name="Subject" type="xs:string" minOccurs="1"
maxOccurs="1"/>
      <xs:element name="Player" type="tPlayer" minOccurs="0"
maxOccurs="unbounded"/>
```

```
      <xs:element name="Weapon" type="tWeapon" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="ActionStatus" type="tActionStatus" minOccurs="0"
maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>

<xs:complexType name="tPlayer">
    <xs:sequence>
      <xs:element name="Money" type="xs:integer"/>
      <xs:element name="Life" type="xs:integer"/>
      <xs:element name="Weaponry" type="tWeaponry"/>
      <xs:element name="Password" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required" />
  </xs:complexType>

<xs:complexType name="tWeapon">
    <xs:sequence>
      <xs:element name="Life" type="xs:integer"/>
      <xs:element name="Power" type="xs:integer"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required" />
  </xs:complexType>

<xs:complexType name="tWeaponry">
    <xs:sequence>
      <xs:element name="Weapon" type="tWeapon" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>


<xs:complexType name="tActionStatus">
    <xs:sequence>
      <xs:element name="OwnDamage" type="xs:integer" minOccurs="0"/>
      <xs:element name="OpponentDamage" type="xs:integer" minOccurs="0"/>
      <xs:element name="WeaponDamage" type="xs:integer" minOccurs="0"/>
      <xs:element name="Bounty" type="xs:integer" minOccurs="0"/>
      <xs:element name="ServerMessage" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
```

Explicação: o conteúdo da mensagem `Content` tem um assunto `Subject` do tipo cadeia de caracteres `xs:string` que esclarece o propósito da mensagem. O elemento ActionStatus está reservado para as respostas do servidor pelo que não devem ser escritos dados no mesmo.

- No caso da utilização do método `public string GetPlayerList(string xmlFormatedMessage, string password)` apenas o campo `Subject` deve ser preenchido. O servidor responde com o estado da operação no campo `ServerMessage` do elemento `ActionStatus` e devolve varios registos `Player` (um para cada adversário) com os repectivos dados.

- No caso da utilização do método `public string Fight(string xmlFormatedMessage, string password)` os campos `Player` e `Weapon` devem conter o jogador que vai ser alvo do ataque e a arma utilizada no

mesmo. O servidor responde no campo `ActionStatus` indicando o dano no jogador atacante `OwnDamage` , no jogador atacado `OpponentDamage`, na arma `WeaponDamage` e uma mensagem com o relatório da operação `ServerMessage`.

- No caso da utilização do `public string GetStatus(string xmlFormatedMessage, string password)` apenas o campo `Subject` deve ser preenchido. O servidor responde com o estado da operação no campo `ServerMessage` do elemento `ActionStatus` e o estado actual do jogador em apenos um registo `Player`.

- No caso da utilização do `public string GetCatalog(string xmlFormatedMessage, string password)` apenas o campo `Subject` deve ser preenchido. O servidor responde com o estado da operação no campo `ServerMessage` do elemento `ActionStatus` e devolve vários registos `Weapon` (um para cada arma disponível) com os repectivos dados.

- No caso da utilização do `public string Buy(string xmlFormatedMessage, string password)` o campo `Subject` e campo `Weapon` devem ser preenchidos. O servidor responde com o estado da operação no campo `ServerMessage` do elemento `ActionStatus` e a arma adquirida `Weapon`.

# 3. IMPLEMENTAÇÃO

## 3.1 Objectivos

1. Implementação de um XML Schema que valide o registo das batalhas em que o jogador esteve envolvido. O registo de batalhas deve obrigatoriamente manter a seguinte informação: um identificador de batalha, o nome do adversário, o dano provocado ao adversário, o dano sofrido, o dano da arma, o relatório do servidor, um "timestamp" com a hora da batalha.
2. Implementação de um XML Schema que valide o registo de todas armas utilizadas. O registo armas deve obrigatoriamente manter a seguinte informação: toda a informação relativa à arma, o número de vezes que a arma foi utilizada.
3. Implementação de uma estrutura de dados que permita carregar e trabalhar os ficheiros e as representações XML em memória e suporte serialização para envio de mensagens e salvar em ficheiro.
4. Implementação de todos os mecanismos de construção de mensagens para comunicar com o servidor.
5. Implemetação do Cliente com uma interface adequada e que mostre o registo de armas e batalhas e permita jogar interagindo com o servidor nos modo manual e automático (a aplicação autonomamente joga maximizando os ganhos).

## 3.2 Ferramentas a Utilizar

Linguagem C# no IDE Microsoft Viual Studio 2005 Professional para o Windows XP Professional SP2.

Biblioteca para interacção com o servidor fornecida pelos docentes através do site da cadeira.

### 3.3 Bibliografia

Fornecida pelos docentes através do site da cadeira.

## 4. RELATÓRIO

O segundo e o terceiro trabalho têm apenas um relatório que deve ser apresentado no final do terceiro trabalho.

## 5. PLANO DE AULAS

**1ª aula:**

Docente:

Apresentação do Trabalho.
Apresentação do IDE.
Introdução aos XML Schemas

Alunos:

Objectivos 1 e 2 e início do 3.

**2ª aula:**

Docente:

Apoio na realização do trabalho

Alunos:

Objectivos 3 e 4 e início do 5.

**3ª aula:**

Docente:

Apoio na realização do trabalho

Alunos:

Conclusão do trabalho.

**Docentes:**

**Teórica:**

José Barata, jab@uninova.pt

**Prática:**

Luís Ribeiro, ldr@uninova.pt
Yves Rybarczyk, yr@uninova.pt