 UNIVERSIDADE DE COIMBRA FACULDADE DE CIÊNCIAS E TECNOLOGIA ***Departamento de Engenharia Informática***	**Assignment #1** **Integração de Sistemas/** **Enterprise Application Integration** **2023/24 – 1st Semester** MEI **Deadline: 2023-10-06**

Notas:

A fraude denota uma grave falta de ética e constitui um comportamento não admissível num estudante do ensino superior e futuro profissional. Qualquer tentativa de fraude pode levar à reprovação na disciplina tanto do facilitador como do prevaricador.

Os trabalhos entregues serão verificados por software de deteção de plágio.

Data Representation and Serialization Formats

Objectives

- Learn text and binary formats for data serialization
- Understand the differences between the most important formats
- Understand the tradeoffs between message size, network transfer time, and serialization and deserialization time
- This assignment will involve data representation formats, such as XML, JSON, or Protocol Buffers.

Resources

Maven:

- Students are advised to use Maven to manage their projects.
- Maven Tutorial: https://www.tutorialspoint.com/maven/

XML:

- XML: http://www.w3schools.com/xml
- JAXB Tutorial – Java.net: https://www.javatpoint.com/jaxb-tutorial

Note: starting on version 9, the Java Architecture for XML Binding (JAXB) is no longer part of the Java Standard Edition. This causes a few problems with the use of this architecture. You may resort to Maven for the rescue: https://stackoverflow.com/questions/43574426/how-to-resolve-java-lang-

noclassdeffounderror-javax-xml-bind-jaxbexception-in-j/46455026. However, you should be aware that version numbers tend to evolve quickly, and you might be looking at an old version number.

- Xalan: http://xml.apache.org/xalan-j/

JSON

- Homepage: https://www.json.org/json-en.html

Protocol Buffers

- Homepage: https://developers.google.com/protocol-buffers/
- Maven dependency: https://mvnrepository.com/artifact/com.google.protobuf/protobuf-java
- Compiler Download: https://github.com/protocolbuffers/protobuf/releases

---

Java XML Training Part (doesn't count for evaluation)

The text suggests examples in Java and XML but students are free to select other languages and data formats. These, however, might not have support from the professors.

Before they start, students may create a Maven project using the following command line, which initializes a "quickstart" archetype (note that this should be a single line):

```
mvn org.apache.maven.plugins:maven-archetype-plugin:3.1.2:generate
-DarchetypeArtifactId="maven-archetype-quickstart"
-DarchetypeGroupId="org.apache.maven.archetypes"
-DarchetypeVersion="1.4" -DgroupId="uc.mei.is"
-DartifactId="simplejaxb"
```

Also, refer to https://mkyong.com/java/jaxb-hello-world-example/ for the dependencies you need to insert in the pom.xml file and for a base example using JAXB.

1. Using JAXB, write the Java classes and the marshalling code that outputs the following XML:

a)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<class>
	<student>
		<name>Alberto</name>
		<age>21</age>
	</student>
	<student>
		<name>Patricia</name>
		<age>22</age>
	</student>
```

```
        <student>
                <name>Luis</name>
                <age>21</age>
        </student>
</class>
```

b)

```
<?xml version="1.0" encoding="UTF-8"?>
<class>
        <student id="201134441110">
                <name>Alberto</name>
                <age>21</age>
        </student>
        <student id="201134441116">
                <name>Patricia</name>
                <age>22</age>
        </student>
    <student id="201134441210">
                <name>Luis</name>
                <age>21</age>
        </student>
</class>
```

c) (An XML equivalent to this is also acceptable)

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated automatically. Don't change it. -->
<class xmlns="http://www.dei.uc.pt/EAI">
  <student xmlns="" id="201134441110">
    <name>Alberto</name>
    <age>21</age>
  </student>
  <student xmlns="" id="201134441116">
    <name>Patricia</name>
    <age>21</age>
  </student>
  <student xmlns="" id="201134441210">
    <name>Luis</name>
    <age>21</age>
  </student>
</class>
```

d)

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="test.xsl"?>
<!-- Generated automatically. Don't change it. -->
<h:class xmlns:h="http://www.dei.uc.pt/EAI">
  <h:student id="201134441110">
    <name>Alberto</name>
    <age>21</age>
  </h:student>
  <h:student id="201134441116">
    <name>Patricia</name>
    <age>21</age>
```

```
  </h:student>
  <h:student id="201134441210">
    <name>Luis</name>
    <age>21</age>
  </h:student>
</h:class>
```

e)

```
<?xml version="1.0" encoding="UTF-8"?>
<class>
    <student id="201134441110">Alberto</student>
    <student id="201134441116">Patricia</student>
    <student id="201134441210">Luis</student>
</class>
```

2. Use an online tool or install a tool like *trang* to automatically produce the XSD for the following XML. Change the XSD, to ensure that <direction> can only be one of "dgsg|boinc" or "dgsg|xtremweb", while <timestamp> must be positive. Note that you should always check if the tool inferred the correct schema, or if it requires some manual adjustment.

```
<?xml version="1.0" encoding="UTF-8"?>
<report timestamp="1308046204104" timezone="GMT" version="1.1">
<metric_data>
    <metric_name>cpus_available</metric_name>
    <timestamp>1308046204003</timestamp>
    <value>0.0</value>
    <type>uint32</type>
    <units>cpus</units>
    <spoof>EDGITest|fusion:EDGITest|fusion</spoof>
    <direction>dgsg|boinc</direction>
  </metric_data>
  <metric_data>
    <metric_name>gflops</metric_name>
    <timestamp>1308046204056</timestamp>
    <value>0.0</value>
    <type>float</type>
    <units>gflops</units>
    <spoof>EDGITest|fusion:EDGITest|fusion</spoof>
    <direction>dgsg|boinc</direction>
  </metric_data>
  <metric_data>
    <metric_name>past_workunits</metric_name>
    <timestamp>1308046204058</timestamp>
    <value>0.0</value>
    <type>uint32</type>
    <units>wus</units>
    <spoof>EDGITest|fusion:EDGITest|fusion</spoof>
    <direction>dgsg|boinc</direction>
  </metric_data>
  <metric_data>
    <metric_name>waiting_workunits</metric_name>
```

```
    <timestamp>1308046204059</timestamp>
    <value>0.0</value>
    <type>uint32</type>
    <units>wus</units>
    <spoof>EDGITest|dsp:EDGITest|dsp</spoof>
    <direction>dgsg|boinc</direction>
  </metric_data>
  <metric_data>
    <metric_name>success_rate</metric_name>
    <timestamp>1308046204061</timestamp>
    <value>1.0</value>
    <type>float</type>
    <units>percentage</units>
    <spoof>EDGITest|dsp:EDGITest|dsp</spoof>
    <direction>dgsg|boinc</direction>
  </metric_data>
  <metric_data>
    <metric_name>past_workunits_24_hours</metric_name>
    <timestamp>1308046204064</timestamp>
    <value>0.0</value>
    <type>uint32</type>
    <units>wus</units>
    <spoof>EDGITest|fusion:EDGITest|fusion</spoof>
    <direction>dgsg|boinc</direction>
  </metric_data>
  <metric_data>
    <metric_name>cpus_available</metric_name>
    <timestamp>1308046204066</timestamp>
    <value>0.0</value>
    <type>uint32</type>
    <units>cpus</units>
    <spoof>EDGITest|dsp:EDGITest|dsp</spoof>
    <direction>dgsg|boinc</direction>
  </metric_data>
  <metric_data>
    <metric_name>success_rate</metric_name>
    <timestamp>1308046204067</timestamp>
    <value>1.0</value>
    <type>float</type>
    <units>percentage</units>
    <spoof>EDGITest|fusion:EDGITest|fusion</spoof>
    <direction>dgsg|boinc</direction>
  </metric_data>
  <metric_data>
    <metric_name>gflops</metric_name>
    <timestamp>1308046204092</timestamp>
    <value>0.0</value>
    <type>float</type>
    <units>gflops</units>
    <spoof>EDGITest|dsp:EDGITest|dsp</spoof>
    <direction>dgsg|boinc</direction>
  </metric_data>
</report>
```

3. Now, create an XML file with a catalog of books and use XSLT to display the books in an HTML table. A simple way of accomplishing this task is by means of using Xalan-Java, referenced above.

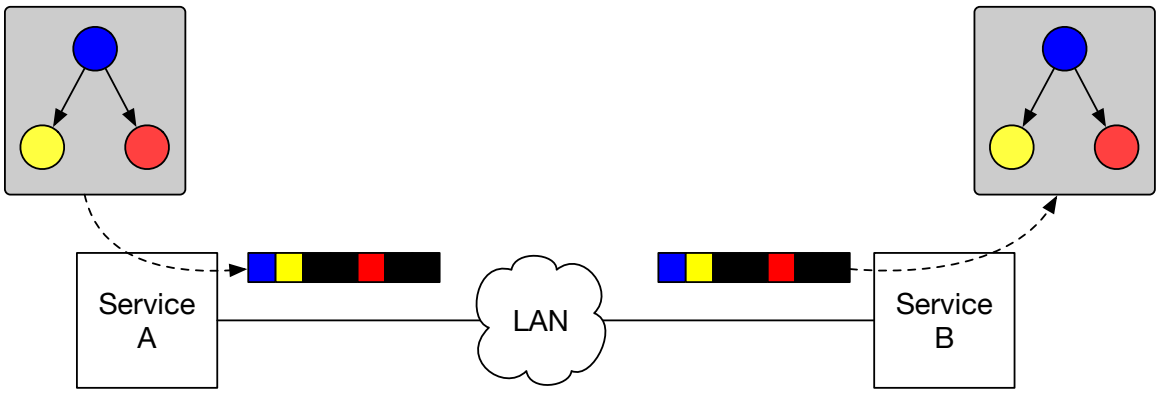

Figure 1 - Serialization, transmission, and deserialization.

Description of the Assignment (for Evaluation)

In this assignment, students should write a small report formatted and presented as if it were a scientific paper on the topic of data representation. For this, they should 1) select two or more data representation formats, 2) elaborate a hypothesis for a microservices scenario, 3) test their hypothesis, and 4) discuss their observation.

For the scenario, students should consider different microservices repeatedly serializing, transmitting, and deserializing data over a Local Area Network. This simulates a setting where multiple different backend services interact towards providing an online service. Students do not need to create any microservice but only to serialized and deserialize (possibly different) data contents repeatedly. Figure 1 shows a simple chain of two microservices.

An example of a hypothesis could be, for example, “JSON is faster than XML” or “Binary formats are faster than text-based formats”. Note that the hypothesis may turn out to be false or partially false, many times depending on the hardware where it is tested. While this is acceptable, students should be careful enough to formulate reasonable hypothesis, which they may discuss with their professors.

Students may use any programming language they wish for their tests if authorized by their professors; they should provide some information in the paper, regarding the language, the data they considered, the sizes of such data, the number of serializations/deserializations they used in the tests, and other conditions they deem to be important. Note that, among other conditions, a good empirical experiment will require rigorous time measurements, multiple trials, and a clear definition of the scope where the results apply. For the sake of reproducibility, students should describe the conditions of the experiments, e.g., computer characteristics, technologies, libraries, versions, etc..

Students must also present and discuss their observations, putting a special emphasis on the explanation of why they got those specific results. For example, if XML turns

out to be faster than JSON in some cases, it would be interesting to learn the exact reasons.

Students may use the following structure for their paper:

Abstract
1. Introduction
2. Concepts
3. Problem Statement
4. Experimental Setup
5. Results
6. Discussion
7. Conclusion

Good work!

Final Delivery

- Students can find a template for the paper here: https://www.overleaf.com/read/fknznfgdbkpt.
- The paper is limited to **3 pages**.
- The paper should be written by groups of **2 students**. We do expect all the members of the group to be fully aware of all the report. Work together!
- Grades will be based on the quality of the paper: how do students describe the data representation formats, the experiment, presentation, and discussion of results; how careful were they while doing the experiments; and how many experiments did they run, to cover the different behaviors of the technologies. Be careful about the way the paper looks.
- Students should deliver the pdf at Inforestudante.
