Bridging Mono/.NET and Java in the SCRIBO Project: The Way to UIMA.NET

François-Régis Chaumartin, Etienne Coumont, Fabio Mancinelli, Olivier Grisel

Bridging Mono/.NET and Java in the SCRIBO Project: The Way to UIMA.NET

François-Régis Chaumartin, Etienne Coumont[1], Fabio Mancinelli[2], and Olivier Grisel[3]

[1] Proxem
{frc,etc}@proxem.com
[2] XWiki
fabio.mancinelli@xwiki.com
[3] Nuxeo
olivier.grisel@nuxeo.com

Abstract. In this paper we introduce the project SCRIBO (Semi-automatic and Collaborative Retrieval of Information Based on Ontologies) and we describe how we have leveraged the UIMA framework in order to integrate existing tools in a general architecture. The paper focuses on how we have bridged the Java and .Net platforms (using the *Mono* framework), describing the problems and an effective solution to make UIMA interoperability possible.

1 Introduction

The SCRIBO project's goal is to develop algorithms and collaborative tools for extracting knowledge from unstructured documents and images. Its distinguishing features rely on the combination of semantic and statistical approaches for natural language processing (NLP) and on its focus on the collaborative dimension of the knowledge extraction process. SCRIBO provides an integrated tool chain for extracting light ontologies from a corpus of documents, for acquiring knowledge by leveraging ontologies, and for extracting structures from digitalized documents. The tool chain will operate in an integrated modular environment built upon standard APIs that facilitates advanced collaboration by providing graphical widgets for annotating semantically evolving texts and images.

One of the main challenges of the SCRIBO project is that of integrating the existing components provided by project's partners and make them interoperate. We chose to leverage the UIMA framework in order to interoperate and to provide components that exposes a standard interface that could be re-used also in context other than SCRIBO itself.

2 Architecture

Figure 1 shows the current architecture of the SCRIBO framework:

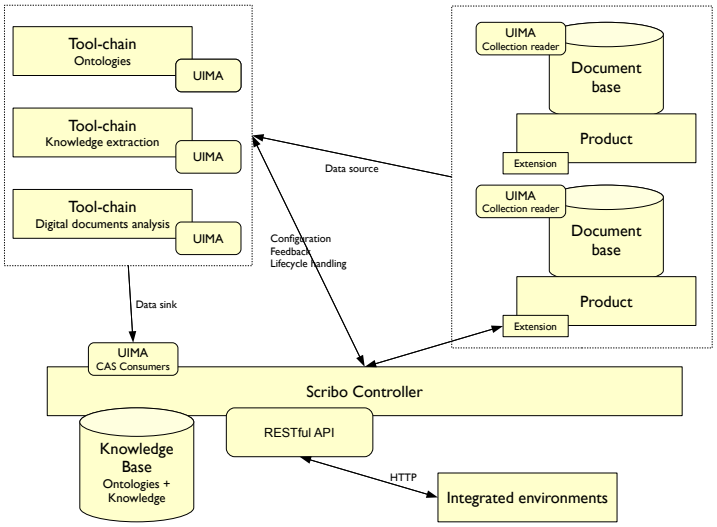

Fig. 1. General architecture

- *Tool-chains.* These are the existing tools that are integrated using the UIMA frameworks. These tools are already existing and developed using different platforms (i.e., Mono (.Net), C++ and a Unix-like integration through Perl and shell scripts), providing the following functionalities:
  - Semi-Automatic extraction and refinement of ontologies.
  - Named entities and relationships extraction.
  - Digital document analysis and partitioning and information extraction.

  These tool-chains are wrapped using the API provided by UIMA framework.
- *Document bases and Products.* These are existing products developed by SCRIBO partners that provides document management facilities and a document base that is used as input to the analysis tool-chains. These document bases are exposed using UIMA collection readers and documents are feed directly to the tool-chains using the UIMA workflow.
- *Controller.* The controller is used to realize a model-view-controller architecture. The controller mediate the interaction between the integrated environments (i.e., the view), the tool-chains and the knowledge base that will store all the information extracted by the analysis (i.e., the model).
- *Integrated environments.* These components provide the user interface to the SCRIBO framework and allow the user interact with it by changing the parameters, visualize the analysis results and modify them collaboratively in order to detect mismatches and correct errors.

3 Bridging Mono/.NET and Java: the way to UIMA.NET

In this section we describe how we succeeded in integrating MONO and .NET-based tools in the UIMA framework, in the context of the SCRIBO project.

Our company, Proxem, has already under development a linguistic platform, Antelope. It includes several Natural Language Processing components (taggers, parsers, semantic, lexicons, Named Entities recognizers...) written mainly with the C# language, and running under the Microsoft .NET Framework or Mono, its open source cross-platform counterpart. Our problem was to integrate them into the UIMA framework, for which only two default implementations (C++ and Java) exist. Rewriting all our components was not something possible; we decided to find a solution to create a UIMA annotator component, running on a MONO or .NET runtime instead of a JVM, and being able to be called from any UIMA client process. First, we tried to create a Web service based on the SOAP standard protocol. Despite much effort, we didn't succeed in this way; there are incompatibility issues between the Web services exposed in UIMA, and their implementation in .NET. Moreover, UIMA Web services expose some classes using Axis libraries, which cannot be easily translated into the .NET equivalent. We then explored a lower-level solution, using sockets to communicate between both virtual machines; we decided to use Vinci, which presents two advantages:

- It is provided as a standard protocol of the UIMA framework.
- It uses only standard Java libraries, so is it easily transposable into .NET.

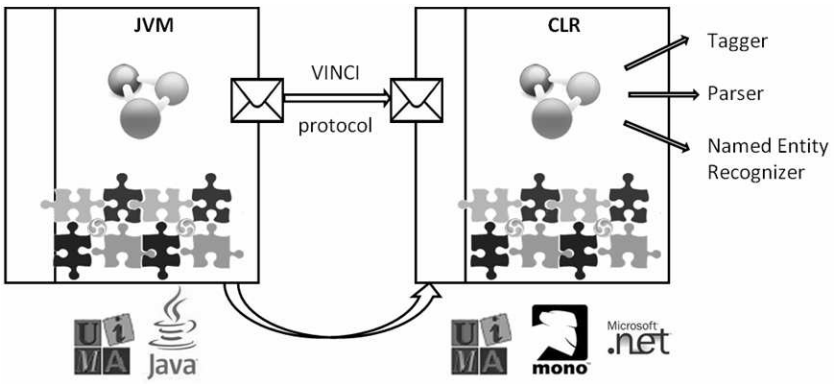

Fig. 2. Architecture of UIMA.NET showing the communication between components.

3.1 Vinci services

Vinci services can be used directly by the Collection Processing Manager (CPM). Vinci handles service naming and location and data transport. Service naming and location are provided by a Vinci Naming Service (VNS). When a Vinci service is started, it has to register itself to the VNS. Then, when a CPM needs to use the service, it has to know the exact name of the wanted service. A query is made to the VNS in order to discover the machine and port hosting the service, which can then be directly invoked.

3.2 Conception of a UIMA annotator for the MONO/.NET Framework

We first had to convert the UIMA libraries (.jar files) into a MONO/.NET assembly. We did it by using the IKVM static compiler, a tool that can translate Java bytecode into MONO/.NET Intermediate Language (the equivalent of Java bytecode). We then created a service launcher, named StartVinciService, a simple MONO/.NET executable which is in charge to call the main function of the Vinci service implementation class, from the library previously translated:

```
VinciAnalysisEngineService_impl.main(args)
```

We also created a new MONO/.NET class library, named AnnotatorLibrary, that contains our annotators. Each annotator is a class that inherits from the UIMA.NET class CasAnnotator_ImplBase, and overrides the process(CAS) method to perform the annotation job, by calling the CAS methods the same way it is done in Java.

This class must be referenced in the annotator XML descriptor file, by specifying its namespace, name, and assembly:

```
<annotatorImplementationName>
  MyNS.MyDotNetAnnotator, AnnotatorLibrary
</annotatorImplementationName>
```

If the assembly is signed, its full name must be specified:

```
<annotatorImplementationName>
  MyNS.MyDotNetAnnotator, AnnotatorLibrary,
  Version=1.0, Culture=neutral, PublicKeyToken=0123abcdef
</annotatorImplementationName>
```

Now we have to create a Vinci deployment descriptor file that references the previous annotator descriptor file as the "resourceSpecifierPath". We are now ready to launch our MONO/.NET Vinci service. We start the Vinci Naming Service if not already done. Then we launch our StartVinciService executable, with one argument, the path to the annotator descriptor file. We can now use this service from a UIMA client, such as the CAS Visual Debugger, which is one of the tools provided with the UIMA Framework.

4 Conclusion

Our first intuition that a SOAP-based standard component should be easy to implement was wrong; in fact, the low-level solution was much simpler to deploy. We demonstrated a fast and easy way to build cross-platforms UIMA applications, and we are proud to provide what we expect to be the first MONO/.NET UIMA implementation.