RC23528 (W0502-098) February 17, 2005
Computer Science

IBM Research Report

Discovering Analysis Components in Unstructured Information Management Architecture

Lev Kozakov, Yurdaer Doganata, Tong-Haing Fin, Youssef Drissi
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Discovering analysis components in unstructured information management architecture

Lev Kozakov, Yurdaer Doganata, Tong-Haing Fin and Youssef Drissi
IBM T.J.Watson Research Center
19 Skyline Drive
Hawthorne, NY, 10532, U.S.A.
kozakov@us.ibm.com

Keywords: unstructured information management, analysis framework, analysis component, component repository

Abstract

Modern intelligence analysis solutions make extensive use of unstructured information management technologies and applications. Developing and deploying application in the area of unstructured information management is costly and requires domain specific knowledge. Existing analysis components are difficult to integrate and reuse due to interoperability problems. In this paper we present the automated component discovery system for analysis components, based on IBM's Unstructured Information Management Architecture. The analysis components are packaged and submitted to the repository for registering and testing. The compliant analysis components are made available for search, download and use. The system facilitates sharing and reusability of analysis components within the architecture framework, as well as planning of analysis solutions and building analysis applications.

1. Introduction

Intelligence analysis requires utilization and integration of many specialized analysis technologies to process the content which may or may not have a structure. These technologies include statistical and rule-based Natural Language Processing (NLP), Information Retrieval (IR), Machine Learning, Ontologies, Automated Reasoning and Knowledge Sources. Solutions to intelligence analysis problems are generally developed by highly specialized scientists and engineers in different platforms and computational environments (see, for instance, [6],[5]). In many cases, these solutions are built independently by using different incompatible techniques which makes the integration highly costly and reusability very difficult.

Although there has been a considerable effort in building analysis components by many domain experts and scientist, the techniques that would enable reusability through discovery of existing components are not extensively used. One of the reasons why an effective analysis component discovery system has not been available was the lack of a common framework where the interoperability of compliant components is ensured. The release of IBM's Unstructured Information Management Architecture (UIMA) made such a component discovery system possible and significant for rapid development of intelligence analysis solutions. The main characteristics of UIMA are outlined in the next section.

In order to enable the reusability and help rapid development of intelligence analysis solutions, a component repository and discovery system has been built for UIMA. This system, which allows component submission, registry, discovery and testing, is the focus of this paper.

One of the challenges of such as a system is to create a set of utilities that facilitate the packaging of software artifacts into a single file with the capability of being deployed on a remote machine. Once the analytics are packaged into components, another challenge is to set the run-time requirements of these components after they are deployed and integrated into a bigger solution. In this paper the necessary steps to ensure reusability of UIMA components is discussed and the implementation of our UIMA component repository solution is presented.

Providing access to a repository of UIMA components and enable component discovery through search and navigation opens up the possibility of building applications through semantic declarations. The solution developers determine the functions that they need to accomplish the task. Then, they use the interface to search the component repository to discover the available components that they could use to build the solution. Designing a solution builder out of available components is not within the scope of this paper but considered as part of the future work.

In a more general context, component-based software development (CBSD) concept focusing on building large software systems by integrating previously-existing software components has been a challenge since 87 (see [1]). The effectiveness of existing approaches have been limited due to lack of mature technologies, standard application programming interfaces, architectural guidelines and real time performance guarantees. In the area of unstructured information management, the availability of UIMA is expected to overcome the difficulties encountered generally in CBSD systems.

In the next sections, we will review UIMA and then outline the major concepts behind the automated component discovery system for UIMA compliant analysis components. Then, we will describe in details the realization of each of the concepts in our system. At the end, we will briefly summarize the benefits of this system, as well as the future work.

1.2. Background

IBM's Unstructured Information Management Architecture (UIMA) is a software framework that was designed to enable the integration of analysis components (see [3],[7]). It supports creating; composing and deploying a broad range of analysis capabilities and linking them to structured information services such as database and search engines services. Rapid integration across technologies and platforms is made possible by employing a variety of different deployment options ranging from tightly-coupled to highly parallel and fully distributed.

The interoperability of the analysis components in a solution is accomplished by using a common analysis structure (CAS) where each component reads from and writes to by using by using published API. The CAS is the primary data structure which UIMA analysis components use to represent and share analysis results (see [4], [8] - chapter 20). It contains:

- The artifact - this is the object being analyzed such as a text document or audio or video stream. The CAS projects one or more views of the artifact. Each view is referred to as a Subject of Analysis.
- A type system description – indicating the types, subtypes, and their features.
- Analysis metadata – "standoff" annotations describing the artifact or a region of the artifact.
- An index repository to support efficient access to and iteration over the results of analysis.

UIMA Architecture defines the following high level components, namely Collection readers, CAS initializers and CAS processors (see [8] - chapter 5). Collection readers read documents from some source, for example a file system or a database. Each document is returned as a CAS that may then be processed by CAS processors. CAS initilalizers further process the documents and populate a CAS from a raw document. For example, if the document is HTML, a CAS Initializer might store a detagged version of the document in the CAS and also create inline annotations derived from the tags. For example, *<p>* tags might be translated into inline Paragraph annotations in the CAS. CAS Processors take a CAS as input and return a CAS as output. There are two types of CAS Processors: Analysis Engines and CAS Consumers. An Analysis Engine is a program that analyzes artifacts (e.g. documents) and infers information about them, and which implements the UIMA Analysis Engine interface Specification. CAS Consumers receive each CAS in the collection after it has been processed by an Analysis Engines. The CAS Consumer may then perform collection-level analysis and construct an application-specific aggregate data structure.

The objects used to store the results of the analysis are called types and the collection of related types form type systems and they are often shared across the analysis engines. Types are reusable assets that can be imported by the developers into their new components.

The UIMA applications are composed by using these UIMA components as the building blocks whose interoperability are assured by UIMA APIs. This enables rapid application development where UIMA compliant solution modules can be plugged-in to build a solution.

In an environment where components are shared and reused, there is a need for a UIMA component repository where different users can browse, search, submit, test and make UIMA components available for others to use. The repository provides for a list of tested components for the solution developers and saves them a great deal of development and testing time.

2. Conceptual system overview

The automated component discovery system for UIMA compliant analysis components is based on mainly the following three concepts:

A. ***Packaging the component code and its resources***
The component code and the resources are packaged into Processing Engine ARchive (PEAR) file, which is then used for storing in the repository, downloading and installing into a local environment and distribution. The structure of PEAR accommodates some mandatory descriptor files along with user files. PEAR packaging enables automated installation and verification of UIMA compliant analysis components.

B. ***Searchable repository for analysis components.***
PEAR files, encapsulating analysis components, are submitted by their developers to a centralized repository. Each component has a unique ID, specified in its PEAR file and enforced by the repository. When a PEAR file is received, the encapsulated component code and its resources are automatically installed in the local repository environment to verify the com-

ponent serviceability. Based on the results of this verification, the submitted component can be registered by the repository. The repository provides a wide range of search and discovery operations based on the metadata associated with registered components. In addition, developers can test their components in the repository environment, using standard document collections provided by the repository or their own collections. The repository provides both interactive and automatic testing capabilities. The testing result reports can be published in the repository as the basis for component rating.

C. ***Deploying and running components***
Analysis applications, running in the UIMA environment, communicate with one or more analysis components using unified UIMA framework API. However, each analysis component may require its own run-time environment configuration, i.e. environment settings like CLASSPATH, PATH, etc. The run-time environment configuration, required by the analysis component, is specified in the PEAR package that encapsulates the component code and resources. The system provides the application container to configure the run-time environment for deploying and running analysis components in UIMA compliant applications.

In the next three sections we discuss in details the realization of these concepts, starting from the PEAR packaging and continuing to the searchable component repository and to the process of deploying and running analysis components. The general architecture diagram of the automated component discovery system is shown in Figure 2.

3. Packaging and installing components

PEAR (Processing Engine ARchive) is the packaging standard for UIMA analysis components. PEAR packages are used for storing and distribution of analysis components, enabling sharing and reuse among other UIMA components or applications. A PEAR package encapsulates the component code and resources required to instantiate and run the component automatically. It also contains several elements that specify the component metadata required for registering, searching, installing, deploying and running the component in the UIMA framework environment. The PEAR package also allows applications and tools to automatically configure runtime environment for verification, deployment and invocation of the encapsulated UIMA analysis component.

The PEAR package is a structured tree of folders and files, including the following elements: The *metadata* folder and some optional folders. The metadata folder contains the PEAR installation descriptor which specifies the information required to install and run the component and the properties files. The PEAR properties files specify the metadata required to register and search the component. The PEAR also contains descriptor files, source codes, binaries, configuration files, data files, other user-defined files and resources. Three deployment types can be specified for the analysis component within the PEAR as follows:

*Standard deployment: T*he analysis component is installed in the local environment; the component instance is deployed within the application process.

*Service type deployment: T*he analysis component is installed in the local environment; the component instance is deployed as a UIMA supported local service; the component deployment is done within the application process.

Network type deployment: The analysis component is installed in a remote environment; the component instance is deployed as a UIMA supported network service; the component deployment is done by the component owner). In this case, the PEAR package does not have to contain files required for component installation, but must contain the network component descriptor (see [8] - sections 4.1.4, 6.5).

Developers create their PEAR packages using the PEAR Packager Eclipse plugin tool (see[8] - Chapter 9). Once the package structure is created, the tool can assist in populating the package with the analysis component files. The PEAR Packager provides a convenient multi-step wizard GUI for creating and editing the PEAR installation descriptor. At the end of the process, the wizard creates the PEAR file and provides an easy way to submit this file to the centralized component repository.

As described earlier in the paper, PEAR packaging enables fully automated installation of the encapsulated analysis components. This means that once a PEAR file is received, the encapsulated analysis component may be automatically installed in the local environment. It is similar to installing J2EE applications from WAR files, except for the fact that analysis components are not limited to Java code, and PEAR packaging provides wider selection of deployment options.

The PEAR installation starts from unpacking the PEAR file in a given directory. Then, the installer loads the XML installation descriptor and checks the local environment (OS, installed toolkits) against the specifications. At the next step, the installer looks at the specifications of required delegate components and other external resources and makes sure all the external references are satisfied. This step may require communications with the component repository in order to automatically download and install required resources. At the last step, the installer performs the verification of component serviceability by creating an instance of the analysis component using the UIMA API. The PEAR installer module is available both as a standalone GUI application (see [8] - chapter 10) and a Java library for use in applications.

4. Using component repository

PEAR files encapsulating UIMA analysis components, submitted by different groups of developers, are stored in the centralized component repository. The repository, at the first place, plays traditional roles of a component warehouse and directory where developers of UIMA analysis components can register their creations and discover creations of others. Once a PEAR file, encapsulating certain analysis component, is received, the repository extracts the component metadata and registers the component. At the next step, the PEAR package is installed in the repository environment to verify the component serviceability. If the installation and verification steps are completed successfully, the repository formally 'accepts' the submitted component, otherwise the repository 'rejects' the component, and the author is notified of the installation or verification errors.

The search and discovery capabilities of the repository are based both on the component metadata included in the *metadata* folder of the PEAR file (like component ID, category, description, owner name, etc.) and on the component specifications, especially its type system and capabilities, containing in the UIMA descriptors. Type systems (see [8] - sections 4.6.3, 4.6.4, 2.3.2) play the central role in building the component repository. The repository backend service extracts component type system information from submitted PEAR files and generates a hierarchical type system tree (see Figure 1) that helps users to get orientation in the variety of available components. The repository provides component discovery services based on available analysis type systems, as well as component capabilities, namely input and output analysis types. In addition to this, the repository allows developers importing registered analysis types and type systems as reusable assets.

Another important feature of the automated component repository is its open component testing and evaluation lab. The system supports both interactive and batch component testing and evaluation modes. In the interactive mode, the developers can run their analysis components, accepted by the repository, using one of the common repository clients described later in this section. In the batch mode, the developers specify the desired test configuration, and submit the specified test configuration to the component testing and evaluation lab. The testing lab installs all analysis components, specified in the test configuration, and performs the required test. The system provides various document collections along with a set of auxiliary UIMA components like collection readers and CAS consumers that can be effectively used for testing and evaluation of analysis components. The developers also can use their own document collections, collection readers and CAS consumers, if available resources do not satisfy their needs.

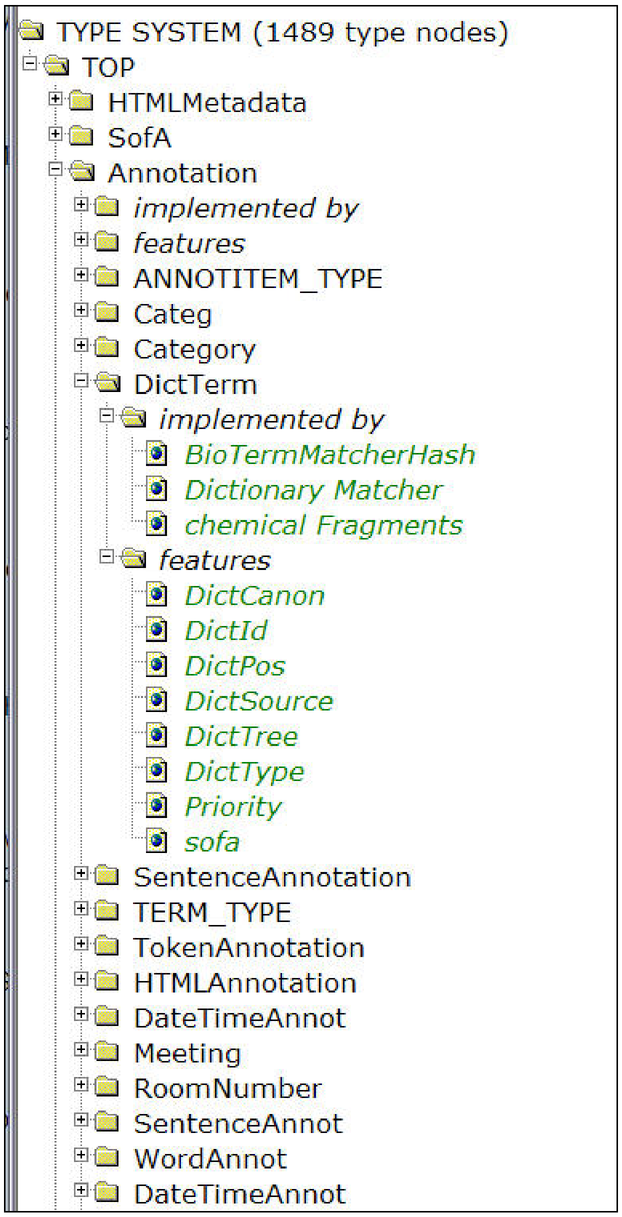

Figure 1. Type system tree fragment

The testing and evaluation lab module utilizes UIMA Collection Processing Engine or CPE (see [8] - chapter 5) to run the required test. The CPE configuration is generated automatically based on the specified test configuration as well as the component specific deployment and run-time configuration parameters, specified in the PEAR installation descriptor. During the test, the system uses the CPE API to collect important run-time statistics, like component throughput, number of documents processed, number of failures, etc.

The testing and evaluation lab module utilizes UIMA Collection Processing Engine or CPE (see [8] - chapter 5) to run the required test. The CPE configuration is generated automatically based on the specified test configuration as well as the component specific deployment and run-time configuration parameters, specified in the PEAR installation descriptor. During the test, the system uses the CPE API to collect important run-time statistics, like component throughput, number of documents processed, number of failures, etc. The test outcome along with the collected statistics is included in the component test report card that may be published in the repository DB and sent back to the component owner. Component test report cards may be used for component rating and comparison of different analysis components.

As soon as more components and type systems are registered and available, finding appropriate components with the right capabilities becomes a difficult task. The automated repository and discovery system has a client/server-based architecture and supports web access. It provides two different kinds of clients for accessing the components and the type systems in the repository: the web browser-based client and the Eclipse-based client. The browser-based client is suitable for users who would like to access the repository without installing any software. The Eclipse-based client is more oriented to developers who need more functions to be used with their Eclipse IDE (see [2]). With the repository clients, users can register, view, and discover components and type systems as described in the previous section. To support various repository clients, the repository provides a common API for accessing and searching the components and type systems. This API makes it possible to implement the repository and discovery system on top of a wide variety of database-structured systems without changing any of the repository's other components.

5. Deploying and running components

UIMA applications use one or more UIMA analysis components to perform desired analysis of input information. Depending on the selected deployment mode (see section 3), required analysis components may be installed in a local environment (i.e. local file system) or in a remote box, and may run in separate processes, as services, or in the same process as the application itself. In all cases, the code of analysis components and their resources are distributed as PEAR files that contain the specifications of the deployment mode and operations (e.g. running specified script for starting a service), as well as the specifications of required run-time environment variables settings. Note, that many analysis components require more than just the traditional CLASSPATH settings, e.g. PATH settings for loading dependent native libraries or other settings for loading resources, such as dictionaries, etc.

The automated component discovery system provides the UIMA application container service for deploying the analysis components from their PEAR packages, configuring the run-time environment and running UIMA compliant applications. The application container reads the specifications in the PEAR installation descriptor and combines them with the default PEAR environment settings, such as adding all JAR files in the lib directory to the CLASSPATH, and adding the bin directory to the PATH. Once the run-time environment is properly configured, the application can instantiate all required analysis components using standard UIMA API. The UIMA application container service is available as Java library that can be used in customer's code.

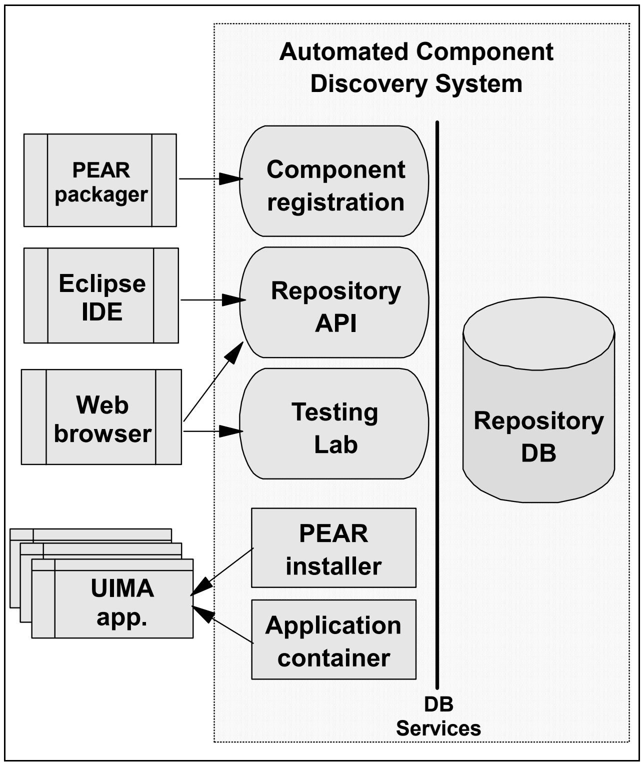

Figure 2. Architecture diagram

6. Conclusion

This paper describes the automated component discovery system built on top of IBM's Unstructured Information

Management Architecture framework and discusses its main concepts and implementation aspects. The main goal of the system is facilitating sharing and re-usability of UIMA analysis components by providing common packaging standard, searchable component repository and testing environment, as well as the run-time application container for UIMA developers. The system provides wide variety of options for discovering analysis components. The component repository back-end module provides an API that connects different clients to the repository services, and enables automated or semi-automated analysis solution planning based on component capabilities. As the UIMA framework is accepted by more developers of unstructured information analysis technologies and applications, the automated component discovery system will play significant role in the mind sharing and consolidating efforts of the developers' community.

7. Future work

The first phase of the automated component discovery system is available for UIMA developers at IBM Research since summer 2004. Our future plans include the following areas:

- implementing API for interacting with the component testing and evaluation lab module;
- integrating all the system tools within the Eclipse based UIMA development environment;
- enabling optimization of analysis solution planning, based on the component rating;
- developing tools for interactive and automated quality evaluation of analysis components;
- building ontologies of type systems and analysis components to enable knowledge based and semantic based planning of analysis solutions;
- developing a semantic based solution builder.

References

[1] Brooks, F. P. Jr. April 1987. No Silver Bullet: Essence and Accidents of Software Engineering. Computer, 20(4).

[2] Eclipse IDE. http://www.eclipse.org

[3] Ferrucci, D.; Lally, A. 2004. Building an example application with the Unstructured Information Management Architecture. IBM Systems Journal, 43(3), pp.455-475.

[4] Götz, T; Suhre, O. 2004. Design and implementation of the UIMA Common Analysis System. IBM Systems Journal, 43(3), pp.476-489.

[5] Hyland, R.; Holland, R.; Clifton, C. 2003. GeoNODE: Visualizing News in Geospatial Context. The MITRE Corporation, http://www.mitre.org/tech/itc/g061/geonode/AFCEA99_GeoNODE_paper.html

[6] Novel Intelligence from Massive Data (NIMD). 2002. ARDA, http://www.ic-arda.org/Novel_Intelligence/

[7] Unstructured Information Management Architecture SDK. December 2004. IBM alphaWorks, http://www.alphaworks.ibm.com/tech/uima

[8] UIMA SDK User's Guide and Reference. December 2004. IBM alphaWorks, http://www.alphaworks.ibm.com/tech/uima