Tjoa, Wagner (Eds.): Proceedings of the Dexa '07, Regensburg
18th International Conference on Database and Expert Systems Applications
ISBN 0-7695-2932-1, pp. 259-263, ©IEEE 2007.

# An MDA Approach to Implement Personal IR Tools

Sven Meyer zu Eissen Benno Stein
*Faculty of Media, Media Systems*
*Bauhaus University Weimar, Germany,*
{*sven.meyer-zu-eissen* | *benno.stein*}*@medien.uni-weimar.de*

## Abstract

*We introduce* TIRA[1]*, a software architecture for the rapid prototyping of tailored information retrieval (IR) tools.* TIRA *allows to compose personal IR tools from atomic IR services, following the model driven architecture (MDA) paradigm: In a first step, an IR process is defined independently from platforms, by means of a UML activity diagram. In a second step, the activity diagram is transformed to a platform specific model, which is executed in a distributed environment.*

*Major driving force behind our research is the question of personalization: We see a large gap between information retrieval theory and algorithms on the one hand and their implementation and deployment to satisfy a personal information need on the other. This gap can be closed with adequate software engineering means, and* TIRA *shall contribute in this respect.*

## 1. Introduction

Information retrieval (IR) is considered as key technology to address the problem of information overload, which is caused by global information accessibility and the increasing number of "information creators". Note that information retrieval is not a universal answer to a generic information need problem but a collective term for myriad solutions to individual information need problems. To become an effective means, retrieval technology must be adapted to personal information needs, which pertains among others to the following points:

1. *Personal Data.* Document sources on which retrieval tasks are carried out include local hard drives, the Web, or intranets.
2. *Personal Preferences.* Typical preferences are language and local settings, or an individual style for result preparation.
3. *Personal Skills.* This characteristic comprises a user's creativity to formulate queries, his/her ability to improve queries iteratively upon search engine feedback, or background knowledge about the retrieval strategies of search engines.
4. *Personal Knowledge.* Even when a personal query formulation skill is highly developed, retrieval success still depends on a user's knowledge of the query domain and the underlying collection (e.g. technical terms). This observation applies especially to closed collections or topic-centered collections in corporate intranets.
5. *Personal IR Tasks.* Advanced personal information needs cannot be suitably addressed with a keyword query approach but require the statement of a tailored IR process. Examples include plagiarism detection, opinion extraction, and filtering according to document quality.

We argue that the current generation of IR tools is not flexible enough to address the above points, especially Point 5. The "course of action" in current IR tools is hardwired, i. e., a user is restricted to specify a query along with a few parameters and cannot adapt or even design the retrieval process itself. We propose an IR software architecture that follows a service composition paradigm: Given an advanced IR problem, a tailored tool that solves this problem shall be constructed by simply selecting and connecting services from a set of "IR building blocks". Our architecture allows to specify and to store personal IR tasks on a user's personal device and to execute these tasks in a distributed environment.

The remainder of this paper is organized as follows. Section 2 relates IR theory to IR software and motivates the service-oriented approach, Section 3 discusses formalisms to specify IR processes, and Section 4 introduces architectural concepts behind TIRA.

[1] Acronym for Text-based Information Retrieval Architecture.

## 2. From IR Theory to IR Software

Depending on the retrieval task a document $d$ can be viewed under different aspects: layout, structural or logical setup, or semantics. A computer representation $\mathbf{d}$ of $d$ must capture the required portions of theses aspects. For this purpose information retrieval theory put forth the necessary underpinning: linguistically motivated retrieval models, algorithms for text analysis, data structures for managing gigabytes, or new statistical insights. Based on these results $\mathbf{d}$ can be designed purposefully, with respect to the structure of a formalized query, $\mathbf{q}$, and also with having a particular retrieval model, $\mathcal{R}$, in mind. $\mathcal{R}$ provides the linguistic rationale for the model formation process behind the mapping $d \mapsto \mathbf{d}$ and provides a concrete means, $\rho(\mathbf{q}, \mathbf{d})$, for quantifying the relevance between a formalized query $\mathbf{q}$ and a document's computer representation $\mathbf{d}$.

The operationalization of future IR processes is far off from being a standard software engineering task, since the retrieval model $\mathcal{R}$ shall be adaptable to personal information needs. Current implementation practice is to maintain software libraries that provide generic IR functionality, and to reuse them in other projects. Although this practice has approved in general settings, the IR process design situation comes with properties that allow for a more powerful modeling perspective:

- IR processes are composed of rather autonomous software building blocks, which are called modules here. Basically, each module provides a service that transforms an input data structure into an output data structure. Examples for such modules include import filters, clustering algorithms, validity measures, ranking functions, classifiers, language taggers, and visualization algorithms.
- Information retrieval theory put forth different solutions for one and the same task or for a class of related tasks.[2] Examples include the different approaches to stemming (statistical algorithms, rule-based algorithms [8]) and to keyword extraction (internal versus external methods, corpus-based methods).
- Several tasks within an IR process are addressed with a parameterizable base algorithm.[3] Examples include the language-specific stemming and stopword filtering [9], which take language-specific rules or word lists as their input.
- IR processes are subject to frequent change: they are optimized, tested with new ideas, and adapted to changing information needs.
- Typically, various parts of an IR process can be executed in parallel, especially when documents are analyzed with respect to different objectives. An example is the intrinsic similarity analysis of a document collection with respect to topic, to genre, as well as to writing style [4, 10].
- A set of standard modules that is useful for virtually any IR process can be identified. Examples include modules for stemming, modules for stopword removal, and conversion modules for binary formats like Adobe Acrobat (PDF) or Microsoft Word.

[2] Observe the connection to the Strategy Design Pattern [2].

[3] Observe the connections to the Factory Design Pattern and the Decorator Pattern [2].

```
Input:   URL u, dictionary dict, stopword list stl.
Output:  genre and topic class for the document at URL u.

Text ht=download(u);
Text plain=removeHTMLTags(ht);
Text filtered=removeStopwords(plain, stl);
Features topicModel=
           buildTopicModel(filtered, dict);

Language lang=detectLanguage(plain);
Features presentF=buildPresentationF(ht);
Features posF=buildPOSF(plain, language);
Features genreModel=union(presentF, posF);

int topicClass=classifyTopic(topicModel);
int genreClass=classifyGenre(genreModel);

return(topicClass, genreClass);
```

**Figure 1. IR process for the sample categorization task, specified in pseudo code.**

These points exhibit the modular nature of IR processes and, in particular, the benefits when this nature is actually exploited. For the design of IR tools we now introduce a two-step procedure: In a first step an IR process is specified in a diagrammed form; in a second step, this specification is automatically instantiated and deployed as a distributed software system.

## 3. Specification of IR Processes

Consider as an example an IR task where a document shall be categorized according to both a given topic taxonomy and a given genre taxonomy. Figure 1 depicts a specification of the underlying IR process in pseudo code: The topic model and the genre model that are constructed from the document found at an URL u form the input for previously built classifiers. Note that several text representations, including HTML text, plain text, and filtered text are necessary to perform the task.

This kind of specification is current practice in a—what we call—*library-based* modeling approach, but it does not take the nature of IR processes into account: (*i*) the replacement of a module entails tedious and error-prone code and data structure replacements, (*ii*) it requires in-depth knowledge concerning the library, (*iii*) the exploitation of the concurrency between particular subtasks leads to an inflexible design since such behavior must be hard-wired in the underlying execution model (in the form of threads or remote function calls), (*iv*) the deployment strategy must be hard-wired as well.

We propagate to specify an IR process at a conceptual level, by means of a diagrammed modeling language. In the past, different modeling tools have been proposed for similar purposes; they can be classified as control flow dominant, data flow dominant, struture-oriented, time-oriented, data-oriented, and hybrid approaches [12].

Most IR processes can be considered as data flow dominant, i. e., they are invoked by a user who asks to process a query, whereas a module can be executed only if its preceding modules have delivered their data. In addition to prescribing data dependencies, a modeling approach for IR processes must allow for defining concurrency (branching and synchronization) since parts of an IR process may be executed in parallel. Moreover, a modeling approach should support explicit typing in order to analyze module composition constraints with respect to input and output parameters. Finally, depending on the modeling granularity, it can be useful to define iterations on parts of an IR process as well as conditions on the produced data. In the following we argue why UML activity diagrams [7] provide adequate modeling capabilities to specify IR processes.

UML activity diagrams combine novel ideas from Web service flow languages like BPEL [1] with traditional concepts like the token concept from Petri nets to specify control flows and data flows between so-called actions. In particular, action nodes, object nodes, and control nodes are connected with directed edges that specify either a data flow or a control flow [3]. Figure 2 shows an activity diagram of the IR process for our sample categorization task.

In UML activity diagrams the action nodes represent tasks, which are software modules in our setting. Object nodes may be placed between action nodes, representing data objects that are transferred between action nodes. Alternatively, connectors, called “pins”, which are attached to the action nodes, can specify the data type that is accepted as input or produced as output by an action node.

Control nodes further divide into decision nodes, merge nodes, fork nodes, and join nodes. Decision nodes delegate control flow exclusively to one of several possible branches, depending on a condition bound to the node; their counterpart are merge nodes. Concurrency is modeled with fork nodes and join nodes, which indicate the concurrent execution and the subsequent synchronization of control flows and data flows. Finally, buffer nodes, which are specialized object nodes, can be used to define a buffering strategy for concurrent processing.

An activity diagram may be partitioned into so-called swimlanes in order to group nodes and edges according to common properties. Such logical groups are oriented at the user-defined semantics (a closed sub-retrieval-task for example) and allow for the structuring of complex IR processes.

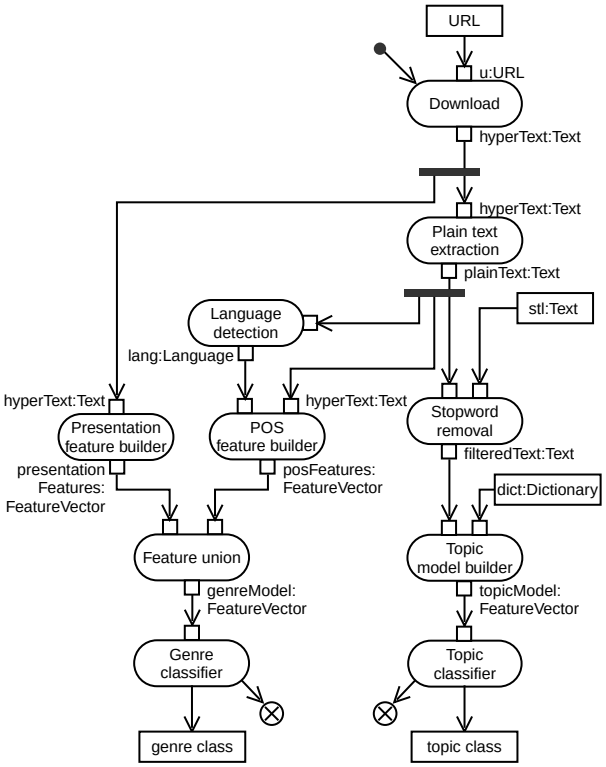


**Figure 2. IR process for the sample categorization task, specified as UML activity diagram.**

**Discussion** Apart from being intuitive, UML activity diagrams are widely accepted as modeling tool. Moreover, advanced concepts that allow for the modeling of data streams, parameter sets, stereotypes, action and time events, exceptions, and exception handlers render this diagram form ideal for our purposes. In the upcoming UML 2.1 specification, conditional nodes and iteration nodes, which remind of block diagram elements, will probably be included, making UML activity diagrams even more intuitive for control flow modeling.

## 4. Operationalizing IR Processes with TIRA

UML activity diagrams are not bound to programming languages, operating systems, middleware, or system architectures. I. e., in terms of the model driven architecture (MDA) paradigm, an IR process specified with a UML activity diagram can be considered as a platform-independent model (PIM) [5]. To make an IR process operable, a target platform has to be chosen, and the PIM must be transformed into an executable platform-specific model (PSM).

In this context a platform denotes the next lower abstraction layer on which a particular model is represented in a more concrete form. For example, J2EE and CORBA are possible platforms for a business process implementation, and, the Java Development Kit in turn is a possible plat-

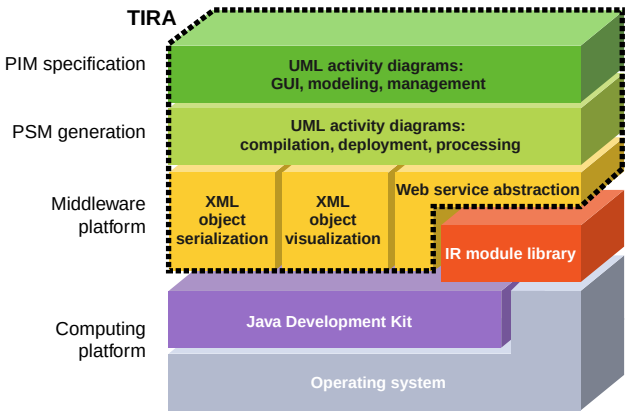


**Figure 3. The layer architecture of TIRA on top of a computing platform. The IR module library is not part of TIRA but provides an extensible container for IR-related algorithms and data structures.**

form for a CORBA implementation. Generally speaking, the transformations along descending platform layers prescribe the path by which a PIM is rendered executable. The OMG denotes a platform that is in-between the PIM and executable code as middleware platform [5].

Just as within other MDA-based application scenarios it is our objective to define, to develop, and to implement the transformation of a PIM to a lower platform layer. However, in contrast to many MDA-based application scenarios we are not interested in the handling of a *variety* of middleware platforms and their related transformations, but in the development of a particular middleware platform that is suited to execute personal information retrieval tasks. Put another way: Our focus is on rapid prototyping, reduced turn-around times, and minimized effort for implementation and test.

Figure 3 shows our implemented proposal for the layer architecture of TIRA: The input PIM is an IR process, modeled as UML activity diagram; a PIM can be compiled and deployed, becoming an executable PSM this way. Modules with the core IR functionality are comprised in an open IR library. The library encapsulates the modules as Web services to make them transparently usable from a PSM via remote function calls. Data objects that are required or produced by the IR modules are materialized as XML objects.

## 4.1. From PIM to PSM

An activity diagram, either loaded from file or modeled interactively with the TIRA GUI, is represented as an object structure in computer memory. The structure is oriented at the UML meta-model [6] and reflects the important elements of activity diagrams, i. e., there are instances of action nodes, fork nodes, etc., which are interconnected by data nodes. The action nodes are bound to IR modules, which in turn are encapsulated as Web services; the data nodes are bound to XML objects.

Figure 4 shows the part of TIRA's class design that models how action nodes are simulated. Instead of modeling an IR module as subclass of the action node class, action nodes are instantiated using a factory class and configured with a symbolic action name. The factory class looks up the URL of the Web service that is associated with the action name at TIRA's service registry and provides the action node instance with this URL. Each IR module that is registered in the service registry comes with a self-description in XML format: input as well as output data types are specified in XML schema. This approach keeps TIRA open, since it allows for registering and executing new IR modules without recompiling TIRA's source.

The object structure is provided with a Petri-net-like token semantics. Simulating the activity diagram means to check the availability of an action node's input data, to allocate processing resources, and to call the corresponding Web service. On delivery of a Web service result, the associated data tokens are propagated in the object structure.

## 4.2. The TIRA Middleware Platform

The functions in the IR module library take objects as input and return new objects. Instead of supplying the Web service stubs with serializations of these objects, parameter passing is realized with the call-by-name paradigm in its most generic form: a parameter must be a URL, pointing to a serialized XML representation of the respective object. This approach comes with the following advantages.

1. When executing an IR process, a client needs not to transfer the intermediate data between two Web service calls; instead, an invoked Web service fetches the data directly from the given URLs, resulting in reduced data transfer costs.

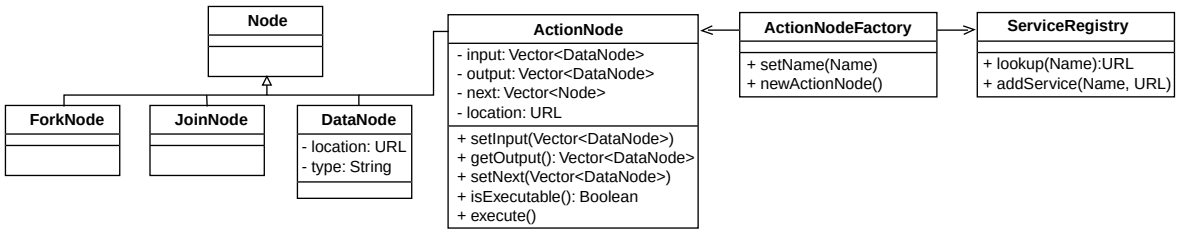


**Figure 4. A part of TIRA's software design given as UML class diagram**

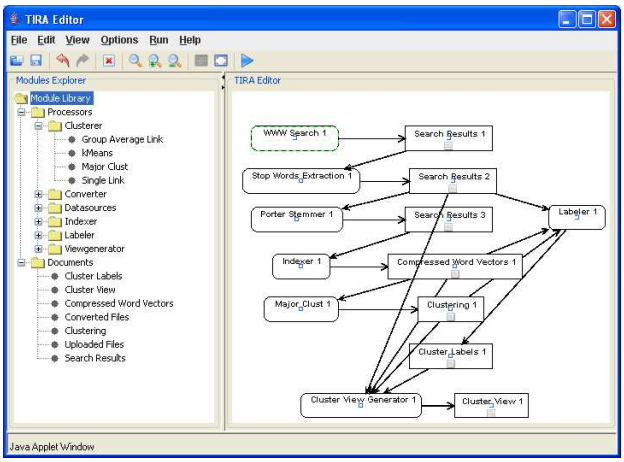


**Figure 5. Process modeling with the TIRA editor.**

2. When two consecutive modules are executed whose corresponding Web services are located on the same server machine, data transfer costs are even lower since the XML files can be directly accessed.
3. The transfer of URL references instead of data objects enables low-bandwidth machines to be fully functional clients. In particular, home users are able to execute personalized IR processes.
4. The use of URLs opens the World Wide Web as address space for data hosting and data sharing.

Because of the broad acceptance of XML the serialization of data objects as XML streams is the means of choice for data exchange. Within TIRA powerful parser generation tools and an XML language binding (JAXB) are responsible for the reading and writing of XML object streams [11].

Intermediate data that is produced during the execution of an IR process is made available for visual inspection, which helps debugging IR processes and lets a user reason about the underlying process. For this purpose the XSL transformation technology is intensively used in TIRA; XSL stylesheets are easy to adapt and to maintain, and they are suited to produce any desired format from the data.

### 4.3. TIRA at Work

Figure 5 shows a screenshot of the TIRA activity diagram editor, which is implemented as a Java applet. The left hand side of the applet shows a selection of available IR modules. A double click instantiates a module, which is then displayed graphically on the right hand side of the Applet. The arrows between the modules display the data flow. A click on a data node invokes the associated XSL transformation, which translates the corresponding data to XHTML and displays the results in a browser.

## 5. Summary

IR processes have reached a ubiquitous presence, be it in the form of search engines at home or at work, on mobile devices or on workstations, or as retrieval components in file systems, document repositories, databases, or knowledge management tools. The reason for this pervasion is the growing information need, the diversity of IR tasks, and the desired degree of personalization. Although specialized retrieval algorithms have been developed in the past, less work has been done on modeling and operationalizing IR processes from a software engineering point of view. This paper contributes to this aspect. Starting from a discussion of modeling approaches for IR processes we introduced TIRA, a flexible MDA solution for the rapid prototyping of tailored IR tools. TIRA allows a user to model an IR process as UML activity diagram, which is automatically transformed into a problem specific model and, based on the TIRA middleware platform, executed by the press of a button.

## References

[1] T. Andrews et al. Business process execution language for web services (bpel4ws) version 1.1. `http://www-128.ibm.com/developerworks/library/specification/ws-bpel/`, May 2003.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley, 1998.

[3] M. Hitz, G. Kappel, E. Kapsammer, and W. Retschitzegger. *UML @ Work*. dpunkt.verlag, 2005.

[4] Sven Meyer zu Eissen and Benno Stein Genre Classification of Web Pages: User Study and Feasibility Analysis. In *KI 2004: Advances in Artificial Intelligence*, Springer, 2004.

[5] Object Management Group (OMG). Model driven architecture (MDA) guide. `http://www.omg.org/docs/omg/03-06-01.pdf`, 2003.

[6] Object Management Group (OMG). The UML metamodel. `http://www.omg.org/cgi-bin/doc?ptc/2004-10-05`, 2003.

[7] Object Management Group (OMG). The unified modeling language (UML) specification, version 2. `http://www.uml.org`, 2005.

[8] M. Porter. An Algorithm for Suffix Stripping. *Program*, 14(3):130–137, 1980.

[9] M. Porter. Snowball. `http://snowball.tartarus.org/`, 2001.

[10] E. Stamatatos, N. Fakotakis, and G. Kokkinakis. Text genre detection using common word frequencies. In *Proceedings of COLING 2000*, Saarbrücken, Germany, 2000.

[11] Sun Microsystems. Java Architecture for XML Binding. `https://jaxb.dev.java.net/`, 2003.

[12] J. Teich. *Digitale Hardware/Software-Systeme*. Springer, 1997.