Journal of Computer Science 1 (1): 83-88, 2005
ISSN 1549-3636
© Science Publications, 2005

Designing a Portlet for Plagiarism Detections Within a Campus Portal

[1]Jinan A.W. Fiaidhi, [2]Zuhoor AlKhanjari, [1]Sabah M.A. Mohammed and [2]Raiya Al-Hinai
[1]Department of Computer Science, Advanced Technology and Academic Centre
Lakehead University, 955 Oliver Road, Thunder Bay, Ontario P7B 5E1, Canada
[2]Department of Computer Science, Sultan Qaboos University, P.O. Box 36
AlKhod 123, Muscat, Sultanate of Oman

Abstract: In an educational context, we are faced with similar challenges. How do we keep the adminis tration, faculty, staff and students well informed about institutional policies and procedures? How do we ensure the student body receives accurate and up-to-date information to help them achieve their educational and career goals? How to check for plagiarism cases? In addition, we hope to build learning communities-communities of students, instructors, administration, faculty and staff all collaborating and constructing strong relationships that provide the foundation for students to achieve their goals with greater success. We also want to promote information sharing so users can build on their experiences at the institution. Plus, we want to provide seamless integration with legacy and other applications in some easy, modifiable and reusable way. One solution to these goals is to provide a support tool for such learning through a learning portal. This portal should provide all users (e.g. students, instructors) with valuable information required. However, building and modifying learning portal is no small task, especially when you consider the shrinking budgets and limited resources in today's economy. Java portlet presents a new solution in which new functionality can be plugged to existing portals. This study shed some light on an ongoing research to develop a plagiarism detection portlet for java student assignments.

Key words: Learning Portal, Portlet, Plagiarism Detection, Software Complexity

INTRODUCTION

Learning portals that encourage collaborative and team-work had been introduced to help overcome the current limitation faced by traditional face-to-face classroom learning approach. The solution is based on a combination of state-of-the-art immersion and interactive technology to induce the learner to non-stop learning experiences and transform the learning environment into a truer-to-life learning environment that is more. Educational studies show clearly, that learning is more fun and more effective in groups and learning portals exactly do that.

Essentially, a learning portal is the doorway to the capabilities provided by a Learning Management System (LMS). There's often a wide variety of courses and content available to the learner via a learning portal. These include everything from courseware to instructor-led classes to synchronous and asynchronous discussion boards and assignment delivery. In many institutions, these resources are drawn from both internal and external resources. This variety is where a learning portal adds its real value. There are various benefits that can be gained out of Learning Portals [1]:

* Accelerates learning, less holdup between learning and action, puts downtime to good use
* Leverages best practice knowledge enterprisewide
* Provide access to all learning opportunities and advice from one place
* Single-source of learning for all functions (esp. important for multi-project workers)
* Provides home base for communities of practice
* Integrates disparate functions, dissolves cognitive boundaries between different functions.
* Moves learning to the learner
* Facilitates "home schooling"
* Resides prominently or the learner's desktop

However, with the growth of such technology it is much easier for plagiarists to copy the materials from each other and put them in their own documents without the permission of the original authors. Indeed facilitators encourage teamwork in doing students studies and exercises. However, they do expect that every student should write the assignments on their own including designs and programs and they insist on testing the acquired skills individually because of its obvious reliability. The solution of plagiarism detection problem cannot simply be solved by having a registered link to cyber-based plagiarism detection packages (e.g. Turnitin,WordCheck, PlagiServe, IntegridGuard, CopyCatch) where mostly they can identify general copying features based on Cut-and-Past material from the internet or from their local databases. Thus, we need a practical approach that can be native to a collaborative environment and can be tailored according to the needs of the learning portal facilitators as well as to be a flexible and extensible service. This article proposes such an approach which can add plagiarism detection facilities to any existing portal using portlets. Generally portals use Java portlets as pluggable user interface components that provide a presentation layer to information systems. It is the next technological step, after servlets and RMI programming in Web application programming.

Structure and Requirements of Java Portlets: By integrating applications and resources, portals let users access information in a simple, straightforward manner. Currently, most portals let users create one or more personal pages composed of portlets-interactive Web mini-applications. Until recently, no standards for portlets exist and thus consuming remote portlets in a generic way or deploying portlets in one portal server that were developed in a different one has been impossible. Two standards released in Fall 2003-the Web Services for Remote Portlets (WSRP) and the Java portlet specification-address these problems.

A portlet is a Java technology based web component, managed by a portlet container, that processes requests and generates dynamic content. Portlets are used by portals as pluggable user interface components that provide a presentation layer to Information Systems. The content generated by a portlet is called a fragment, a piece of markup (e.g., HTML, XHTML, or WML (Wireless Markup Language)) adhering to certain rules. A fragment can be aggregated with other fragments to form a complete document. A portlet's content normally aggregates with the content of other portlets to form the resulting portal page or response [2]. A portlet container manages a portlet's life cycle. The basic portlet life cycle of a Java portlet is:

* Init: initialize the portlet and put the portlet into service
* Handle requests: process different kinds of action- and render-requests
* Destroy: put portlet out of service

The portlet container manages the portlet life cycle and calls the corresponding methods on the portlet interface. Figure 1 shows the typical architecture of a Java portal server that supports WSRP and the Java portlet specification. The portlet container component runs portlet applications conforming to the Java portlet API. The portal Web application component implements the portal use cases, such as sign in, sign up, select portlet layout, aggregate portlet responses and so on. To interact with a portlet, the portal Web application component calls on a specific API provided by the portlet container.

The WSRP producer component provides an implementation of the WSRP interfaces, so other

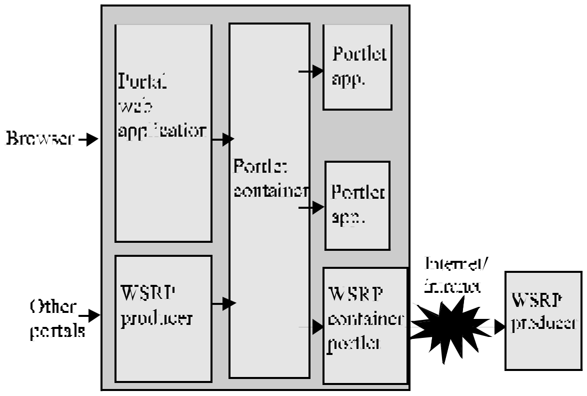


Fig. 1: The Architecture of Java Portlet

consumers can access local portlets. The portal server also provides a WSRP consumer component, implemented as a Java portlet, that acts as a generic proxy of any WSRP producer, allowing the portal to consume remote WSRP portlets. In the WSRP standard, portlet modes refer to the types of functionality a portlet can perform. The standard defines four modes: view, edit, help and preview. It also defines window states, which indicate the amount of space a portlet will be assigned in the page. Window states can be normal, minimized, maximized, or solo. To facilitate the adoption of WSRP and the Java portlet specification, the Apache Software Foundation has launched Jakarta Pluto and WSRP4J. Jakarta Pluto is the reference implementation of a Java portlet container and includes a very simple portal Web application component for testing portlets. WSRP4J builds on Jakarta Pluto, providing a WSRP producer and a proxy portlet of a WSRP producer.

The good thing about Java Portlet that it allows us to package its specification as a .war file for deployment to a J2EE application server. The .war file then can be used to deploy a typical J2EE web application, it contains a WEB-INF/web.xml file to configure the application context. However, with a portlet application, the WEB-INF folder must also contain a portlet.xml file. The portlet.xml file is a descriptor file, containing configuration details about all bundled portlets in the .war file. The following listing shows a simple example of a portlet.xml file. Note how many of the previously-described constructs (portlet mode, preferences, etc.) are defined in this file.

```
<portlet-app>
  <portlet>
   <portlet-name>MyPortlet</portlet-name>
   <portlet-class>com.abc.portlet.MyPortlet</portlet-
class>
   <init-param>--Init param, available in portlet's
PortletConfig instance.
           <name>view-to-present
           <value>/portlet/MyPortlet/startup_view.jsp</v
alue>
```

```
   </init-param>
   <expiration-cache>300</expiration-cache>--Default
expiration for portlet cache (5 minutes)
   <supports>
          <mime-type>text/html</mime-type>--Portlet
supports HTML markup
  <portlet-mode>VIEW</portlet-mode>--MyPortlet
supports modes view and edit
     <portlet-mode>EDIT</portlet-mode>
   </supports>
   <resource-
bundle>com.abc.portlet.MyResourceBundle</resource-
bundle>
   <portlet-preferences>
     <preference>
       <name>Country1</name>      --PortletPreferences
name/value pairs.
       <value>USA</value>
     </preference>
     <preference>
       <name>Country2</name>
       <value>Japan</value>
     </preference>
     --A PreferencesValidator will check any
preferences set.
     <preferences-
validator>com.abc.portlet.validate.CountryValidator
     </preferences-validator>
   </portlet-preferences>
  </portlet>
</portlet-app>
```

The Java Portlet Specification has already been widely adopted by several commercial and open-source vendors. Going forward, portlet developers can take advantage of this standard, thereby insuring compatibility among many different portals.

Linking Java Portlet to Campus Learning Portal: Customizing learning portals is not an easy task without the new Java Portlet. Most of the current learning portals provide administrative applications based on Java technology besides the courseware materials that is likely to be in WebCT. To achieve linkage between the learning portal and the required portlet one need to establish a new channel for providing additional features required by the portlet. In this direction the Portlet Container from the Apache Jakarta project called Pluto (http://portals.apache.org/pluto/) and play the role of this channel. Pluto is the reference implementation of the Java Portlet Specification. The learning portal can embeds the Pluto container and includes a Portlet adapter that makes it possible to install and render Portlets along with its native Channels. The End-User, in this case, can not tell the difference between a Portlet interactions and a Channel interactions. Pluto, when downloaded separately from Apache, is distributed with several components:

* Portlet API (JSR-168)
* Portlet Container
* Portal Driver
* Test Suite Portlet

The learning portal should include the Portlet API, Portlet Container and Test Suite Portlet, but not Pluto's Portal Driver which is not meant to be a fully functional portal. The learning portal acts as the Portal Driver. The learning portal with its implementation of Pluto, requires that the web application context representing the deployed portal web application be set as "cross context" because Pluto dispatches the Portlet request objects to the individual Portlet applications which run in their own web application context. Each container should provide a means to enable this "cross context" setting. This facility can be accomplished using the Tomcat servlet container. Moreover, each portlet can be deployed and linked automatically to ant learing portal using the apachi ant APIs(http://ant.apache.org/). Ant is extended using Java classes. Instead of writing shell commands, the configuration files are XML-based, calling out a target tree where various tasks get executed. Each task is run by an object that implements a particular Task interface. The ant tool takes a Portlet WAR file, rewrites the web.xml file (a requirement of the Pluto Portlet Container) and deploys the results to the servlet container. You can deploy multiple portlets at once. Finally, we may need to publish the new services provided by these portlets. For this purpose, we need to build a Portlet Registry based on all the Portlets that are deployed into the servlet container.

Deploying Plagiarism Detection Portlet for the SQU Learning Portal: The learning portal that we used to deploy our plagiarism portlet is the SQU learning portal used by the Sultan Qaboos University(www.squ.edu.om). However, one can create their own learning portals using the Apatche Portal Driver or by using an open source portal like the uPortal [3]. The SQU Learning portal supports many administrative tasks that can be briefly illustrated in Fig. 2 and 3.
The plagairism detection portlet needs to be deployed in addition to the available functionalities of the SQU Portal (Fig. 4).
AlHinai [4] provides detailed description of the SQU Learning Portal.
The basic technique used for detecting plagairism is to compare the similarity of the text sequences between the given database of documents. Parker and Hamblen [5] defined plagiarized document as a document that has been produced from another document with a small number of texts edit operations but no detailed understanding of the document are required. Many algorithms appeared and among these there are three types of techniques, which are mostly applied to detect plagiarism:

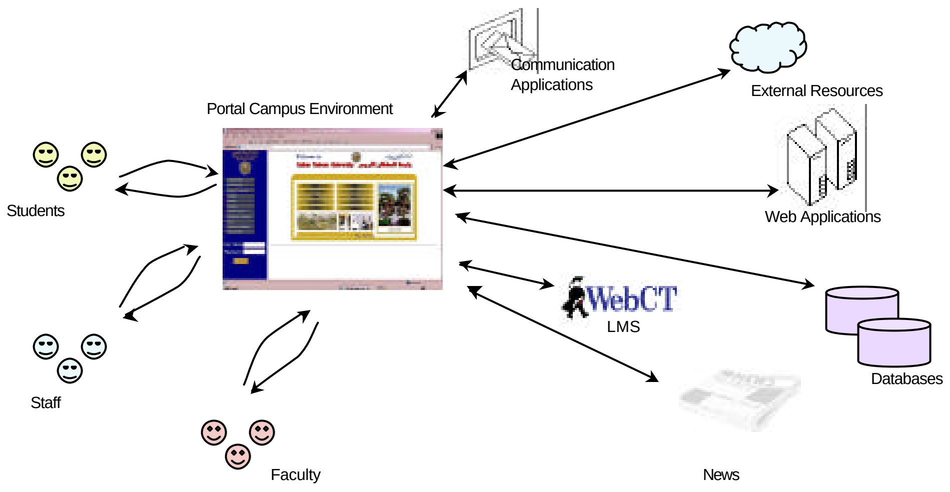


Fig. 2: The General Layout of SQU Learning Portal

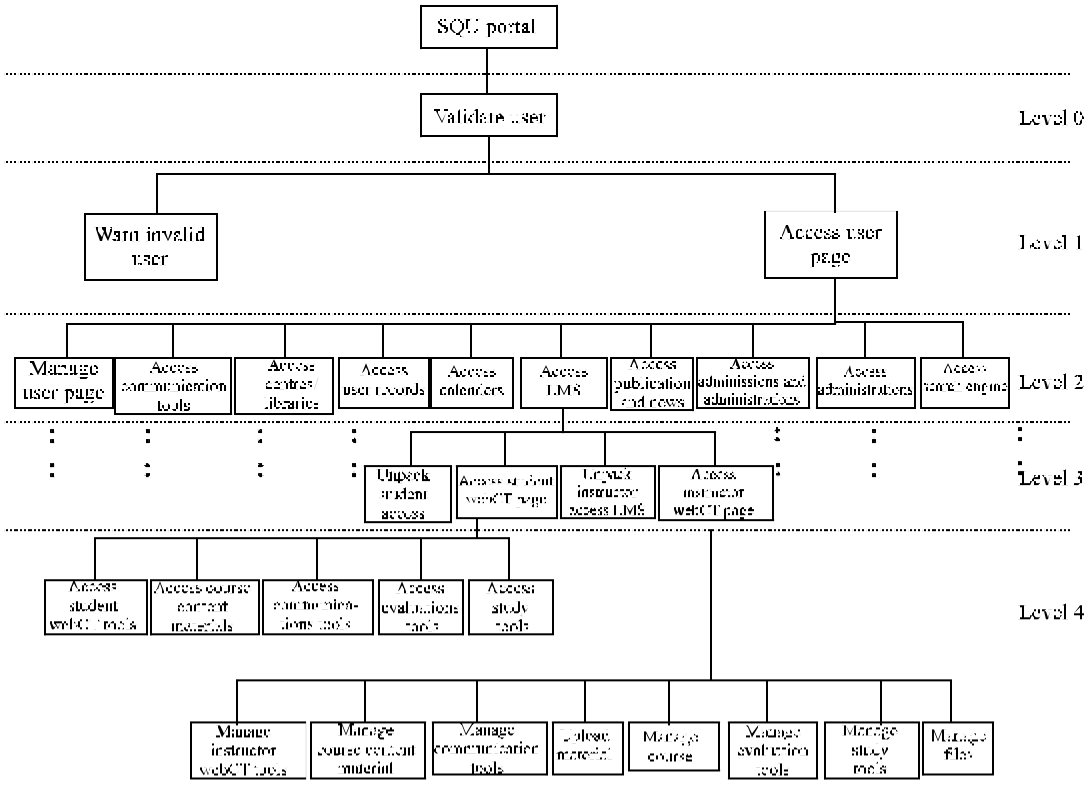

SQU portal
Validate user
Level 0
Warn invalid user
Access user page
Level 1
Manage user page
Access communication tools
Access centres/ libraries
Access user records
Access calendars
Access LMS
Access publication and news
Access admissions and administrations
Access administrators
Access search engine
Level 2
Unpack student access
Access student webCT page
Unpack instructor access LMS
Access instructor webCT page
Level 3
Access student webCT tools
Access course content materials
Access communications tools
Access evaluations tools
Access study tools
Level 4
Manage instructor webCT tools
Manage course content material
Manage communication tools
Upload material
Manage course
Manage evaluation tools
Manage study tools
Manage files

Fig. 3: The Basic Administrative Functionalities Supported by the SQU Learning Portal

* Attribute-counting-metric technique
* Structure-metric/ control-flow technique
* Hybrid technique of both the above technique

The attribute counting technique is the most natural and straightforward technique that is proved to be effective with variety of documents including computer programs [6] as well as it is most natural with Java environment with its reflection API and text paring primitives(e.g StringTokenizer). Attribute-counting-metrics depend on the counting of the frequent occurrences of certain textual features in the document. Thus we attempted to design our plagairism to check the similarity of basically Java Source Documents. We used metrics that are most relevant to Java source documents as well as from those used by other notable systems as MOSS, JPlag and YAP3. The metrics are:

* Unique number of used Java keywords
* Unique number of user methods
* Unique number of interfaces
* Unique number of constructors

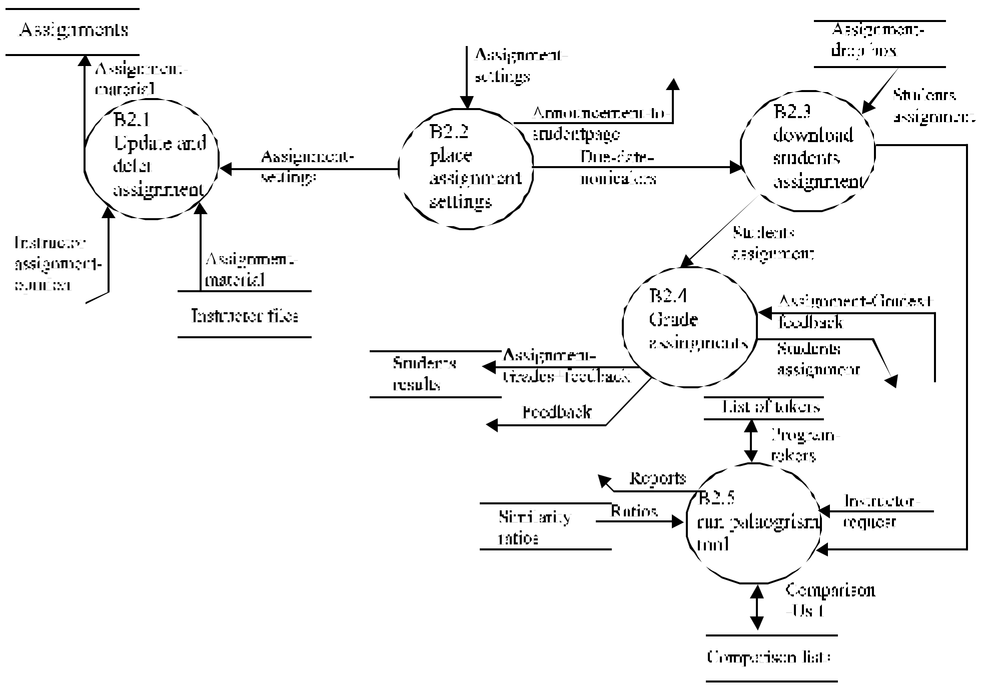
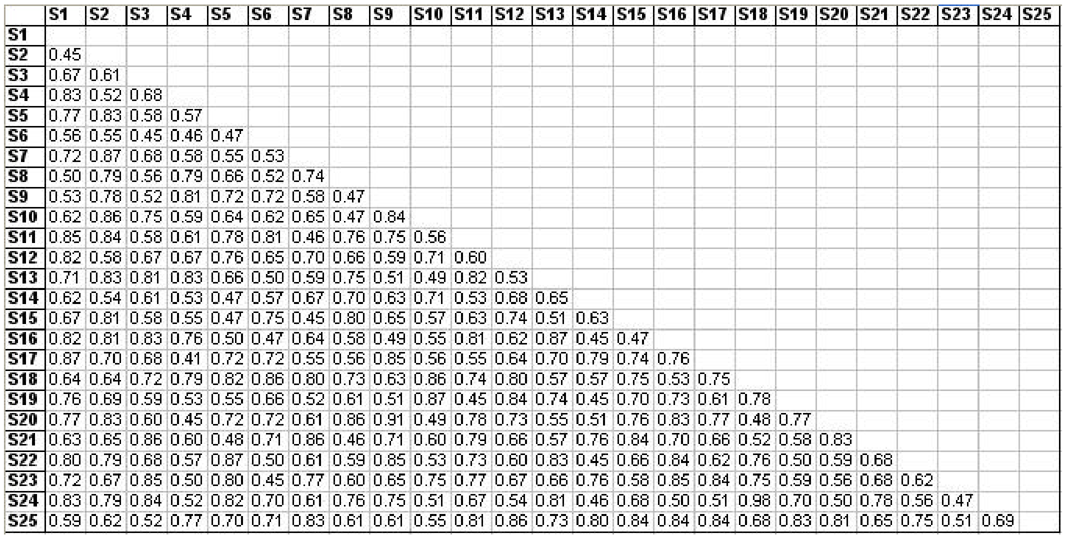


Fig. 4: Adding Plagaorism Detection Functionality to SQU



	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23	S24	S25
S1																									
S2	0.45																								
S3	0.67	0.61																							
S4	0.83	0.52	0.68																						
S5	0.77	0.83	0.58	0.57																					
S6	0.56	0.55	0.45	0.46	0.47																				
S7	0.72	0.87	0.68	0.58	0.55	0.53																			
S8	0.50	0.79	0.56	0.79	0.66	0.52	0.74																		
S9	0.53	0.78	0.52	0.81	0.72	0.72	0.58	0.47																	
S10	0.62	0.86	0.75	0.59	0.64	0.62	0.65	0.47	0.84																
S11	0.85	0.84	0.58	0.61	0.78	0.81	0.46	0.76	0.75	0.56															
S12	0.82	0.58	0.67	0.67	0.76	0.65	0.70	0.66	0.59	0.71	0.60														
S13	0.71	0.83	0.81	0.83	0.66	0.50	0.59	0.75	0.51	0.49	0.82	0.53													
S14	0.62	0.54	0.61	0.53	0.47	0.57	0.67	0.70	0.63	0.71	0.53	0.68	0.65												
S15	0.67	0.81	0.58	0.55	0.47	0.75	0.45	0.80	0.65	0.57	0.63	0.74	0.51	0.63											
S16	0.82	0.81	0.83	0.76	0.50	0.47	0.64	0.58	0.49	0.55	0.81	0.62	0.87	0.45	0.47										
S17	0.87	0.70	0.68	0.41	0.72	0.72	0.55	0.56	0.85	0.56	0.55	0.64	0.70	0.79	0.74	0.76									
S18	0.64	0.64	0.72	0.79	0.82	0.86	0.80	0.73	0.63	0.86	0.74	0.80	0.57	0.57	0.75	0.53	0.75								
S19	0.76	0.69	0.59	0.53	0.55	0.66	0.52	0.61	0.51	0.87	0.45	0.84	0.74	0.45	0.70	0.73	0.61	0.78							
S20	0.77	0.83	0.60	0.45	0.72	0.72	0.61	0.86	0.91	0.49	0.78	0.73	0.55	0.51	0.76	0.83	0.77	0.48	0.77						
S21	0.63	0.65	0.86	0.60	0.48	0.71	0.86	0.46	0.71	0.60	0.79	0.66	0.57	0.76	0.84	0.70	0.66	0.52	0.58	0.83					
S22	0.80	0.79	0.68	0.57	0.87	0.50	0.61	0.59	0.85	0.53	0.73	0.60	0.83	0.45	0.66	0.84	0.62	0.76	0.50	0.59	0.68				
S23	0.72	0.67	0.85	0.50	0.80	0.45	0.77	0.60	0.65	0.75	0.77	0.67	0.66	0.76	0.58	0.85	0.84	0.75	0.59	0.56	0.68	0.62			
S24	0.83	0.79	0.84	0.52	0.82	0.70	0.61	0.76	0.75	0.51	0.67	0.54	0.81	0.46	0.68	0.50	0.51	0.98	0.70	0.50	0.78	0.56	0.47		
S25	0.59	0.62	0.52	0.77	0.70	0.71	0.83	0.61	0.61	0.55	0.81	0.86	0.73	0.80	0.84	0.84	0.84	0.68	0.83	0.81	0.65	0.75	0.51	0.69	

Fig. 5: Comparing 25 Java Students Assignments Using the 11 Metrics

* Total number of used keywords
* Total number of used methods
* Total number of used interfaces
* Total number of used constructors
* Total number of used blocks
* Total number of used Exceptions
* Total number of used Classes

The plagairism portlet uses a version of java parser generated by the Princeton University CPU Java Parser Generator (http://www.cs.princeton. edu/~appel/modern/java /CUP/). These measures are inserted as probs within the generated parser code. The portlet inspects the student assignments at the drop box and performs linear correlation between the metrics extracted from each source. A sample of comparing 25 student assignments in response to the facilitator request of conducting the plagairism test. This test reveals that with a treshold of 0.9 correlation we can identify to suspect cases of plagarism (S20 with S9 and S24 with S18). Indeed changing the threshold may reveal other cases (Fig. 5). One can choose an empirical threshold which can indicate which pairs are likely to indicate suspected cases of plagarism. This is one of the future modifications that we are working on.

CONCLUSION

This study introduced an attempt to design and deplore a plagiarism portlet within an existing campus learning portal. The Java Portlet API is used for this purpose. It provided a new pluggable user interface components that produce a modified presentation layer to learning portal. The plagiarism metrics used are simple Java-based attribute-counting measures. More experiments are required to judge the suitability and accuracy of these metrics. The reusability of the plagiarism portlet is an advantage which can be used with other application such as the deep crawlers used for detecting cyber-plagiarism [7].

REFERENCES

1. Karrer, A., 2000. Building a learning portal. Learning Circuits OnLine Magazine, www.learningcircuits.org.
2. Stefan H. and S. Hesmer, 2003. Introducing the portlet specification. Java World J., http://www.javaworld.com/javaworld/jw-08-2003/jw-0801-portlet_p.html.
3. Brad Rippe, 2003. Start developing portals with JA-SIG uPortal. Java World J., http://www.javaworld.com/javaworld/jw-10-2003/jw-1003-portal_p.html.
4. ALHinai, R.A., 2003. A Plagiarism Detection Tool Supporting Campus Portal. MSc Thesis. Sultan Qaboos University, Oman.
5. Parker, A. and J. Hamblen, 1989. Computer algorithms for plagiarism detection. IEEE Transactions on Education, 3: 2
6. Fiaidhi, J. and S.K. Robinson, 1987. An empirical approach for detecting program similarity and plagiarism within a university programming environment. Intl. J. Computers and Education, 11: 1
7. Fiaidhi, J., S. Mohammed and Z. AlKhanjari, 2003. Designing a vortal for detecting Java programs cyberplagiarism. International Conference on Internet Computing IC03, Las Vegas, USA, June 23-26.