Proceedings in Informatics and Information Technologies

Student Research Conference 2016

Michal Kompan and Pavol Návrat (Eds.)

Keynote by Geraldine Fitzpatrick

STU
FIIT

IIT.SRC 2016: 12th Student Research Conference in Informatics and Information Technologies

Edited by: Michal Kompan and Pavol Návrat
Publisher: Vydavateľstvo STU
Year: 2016
ISBN: 978-80-227-4551-2

Selected papers of the conference not further published

Table of Contents

Transformation from the Heavy Desktop Client to the Lightweight Web Application

Richard BELAN*

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
`xbelanr1@fiit.stuba.sk`

Abstract. Recently the trend is to create applications which are always available online. This leads to expansion of Web technologies and creation of applications directly for Web. With HTML5 and CSS3 there are numerous possibilities to create complex graphical applications on the web. In this paper we propose a way how to migrate heavy graphical applications from desktop to the Internet. This paper also describes the way we took to migrate our graphical desktop prototype and compares some of the current technologies and options available for migration of desktop applications.

1 Introduction

Modelling and designing large, corporate software with complex requirements needs improvements in graphic visualization for easier creation of models, better understanding of diagrams and better collaboration between designers and designer teams through various countries, time zones, departments and divisions in cooperative creation of applications their models and architecture together.

Unified Modelling Language (UML) is a standardized language used in software modelling for describing architecture and functionality of software systems. There are a number of available tools, programs and large applications for creation of UML diagrams with 2D space support. Three-dimensional space brings many new important features for UML diagram creation. Minimization of intersections, reduction of diagram complexity and allowing visualization of complex diagrams in advanced up-to-date graphics for achieving easier to read schemas of large models with decomposing the structure into specific modules, components, objects, types, patterns, authors and time are all of the best benefits of adding another spatial dimension into software modelling and moving UML diagramming from two-dimensional space to three-dimensional space.

With rapid progress in technologies in past years and with everything being on Web, we had to adjust our solution to this growth and migrate our solution from desktop client to a Web Application. This paper brings the journey needed for migrating a graphic application from desktop to web and all the trouble encountered.

* Master degree study programme in field: Software Engineering
Supervisor: Dr. Ivan Polášek, Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies STU in Bratislava

2 Related work

Although there is no commercial three-dimensional UML modelling software, there are some alternatives available. These existing prototypes try to use the third dimension in numerous ways. The three main methods are dividing models and diagrams into layers, using advanced three-dimensional animations [1] and having three-dimensional object representation with possible alternative shapes and forms [3]. H. Koike proposed the first three-dimensional software visualization prototype named VOGUE [1,2], which used third dimension to visualize message sending between processes. Already with adopting UML as a standard for two-dimensional modelling in 1997 many three-dimensional alternatives were proposed. One of the first were J. Gil and S. Kent who first proposed the idea of layers and division of information [4] and they created a three-dimensional notation for UML class diagram and a nested-box diagram [4].

The most complex solution [5] was made by Paul McIntosh and it was his three-dimensional UML state machine diagram. P. McIntosh extensively studied the benefits of third-dimension in software modelling and visualization in comparison with traditional two-dimensional UML approaches [5, 6]. His solution was compatible both with ISO based X3D standards and also with UML standards therefore he named his solution X3D-UML [6]. His approach is best visible on hierarchical state diagrams as his three-dimensional state diagrams uses movable hierarchical layers with ability to apply filtering. Numerous information are not visible on traditional 2D diagrams therefore the person must think outside the diagram. With using third dimension there are new possibilities to show more information on screen and reduce the need of thinking out of diagram [6]. In addition to this research he also studied methodology sequential evaluation used in 3D user interfaces.

Another approach which extensively studied the benefits of third dimension and created a prototype for three-dimensional visualization and software modelling was GEF3D [7]. Their first approach is to visualize connections between two-dimensional diagrams in three-dimensional space [6]. This approach uses the idea of layering information and can be used for model driven development where the models are chained. The second approach is to project diagrams into sides of a cube and arrange connections between diagrams through the cube. GEF3D is so far still in the Eclipse incubator and not released officially.

American vision scientist I. Biederman known for his RBC theory (Recognition-by-components theory) [8] in which he created a set of three-dimensional objects of primitive shape called Geons. He claimed that these objects and composed objects made solely of these objects are easier for human recognition and remembering than any other type of objects. I. Pourang proposed a way of creating diagrams only with Geon shaped objects and also made a study [3] where students remembered better and also could recognize easier the Geon shaped diagrams instead of the standard UML diagrams. Pourang claimed that these diagrams are closer to human perception and by using these shapes the person can recreate the mental map much easier. Although using this method for large, complex diagrams was never proved and tested.

3 Our approach

To stay “up-to-date” with current technological trends and application needs to be always upgraded. There are situations when an additional upgrade is impossible or near impossible and migrating the whole project from one technology to another is a better choice. Now when everything is online and easily accessible there is a need for migrating large desktop client applications to Web. Web offers many advantages opposite to desktop applications. The main advantages are:

- Easier accessibility – Sending access link instead of complete software with installation manuals.
- Smaller client-side client – Compute most of the application logic on backend Server

Considering these advantages and our opportunities in upgrading our existing desktop client we chose to wrap everything usable from our application, reuse it and rewrite it for Web.

3.1 3D-UML desktop client

Our desktop client is a robust graphic application written in C++ and using OGRE3D graphic engine. It is a tool for creating 3D UML diagrams. Currently it supports Sequence, Class and Activity diagrams. Each of the diagrams has their own UML Metamodel created for 3D space like the Metamodel for Class diagram [11] on Figure 1. Metamodels for supported diagrams are compatible with UML standards as they were created from OMG UML Superstructure and Infrastructure [9]. Class diagram supports Force-Directed algorithms [12] like weighted Fruchterman-Rheingold and Sequence diagram is using modern combined fragments for alternative and parallel scenarios [14].

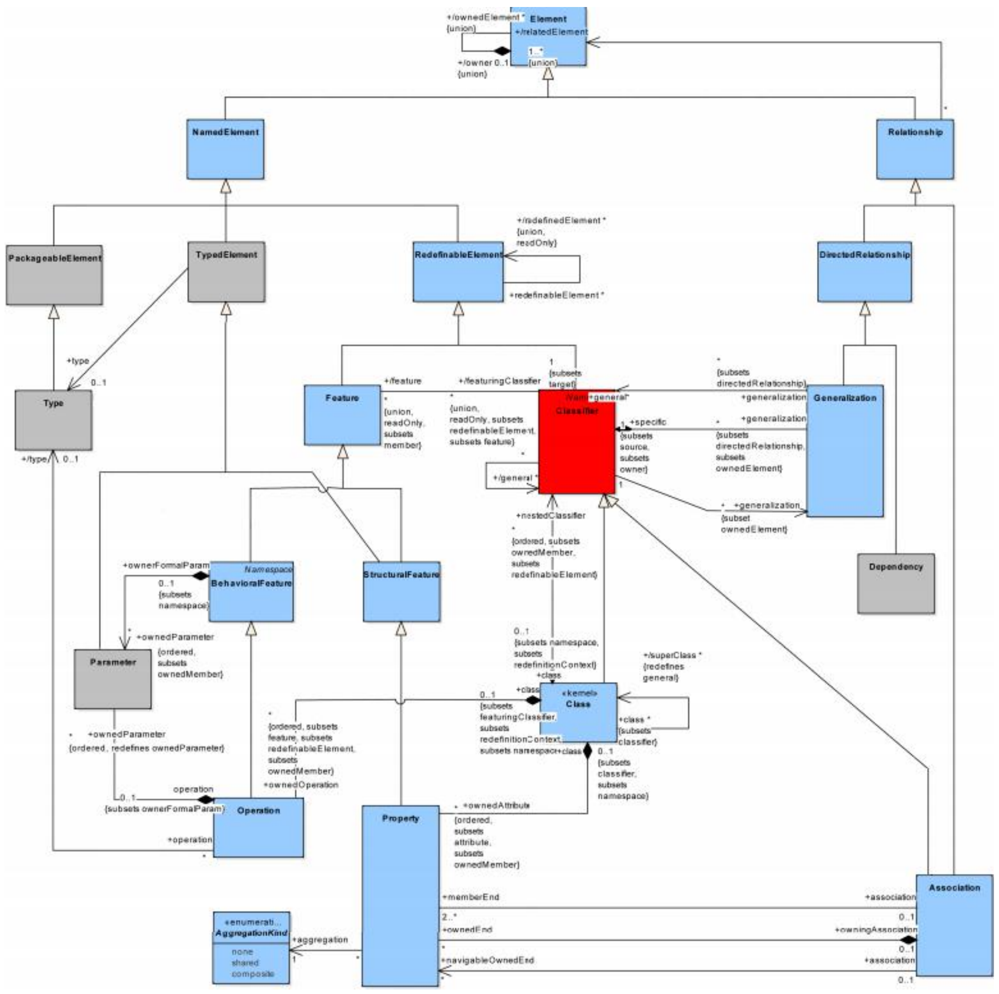


Figure 1. Class diagram Metamodel.

3.2 MMVCC architectural structure

Change of Architectural structure was the first big upgrade to our prototype. The former MVC architectural style was not enough due to our extensive use of 3D elements and backward compatibility with CASE system by saving the elements as 2D too. This way we ended up having two models, one for the 2D elements and another for 3D elements. This was the first part of our transformation from MVC to MMVCC. The second part consisted of dividing the controller. We divided the universal controller for both 2D and 3D elements into two separate controllers. That led to finishing our new architectural style. MMVCC stands for Model-Model-View-Controller-Controller.

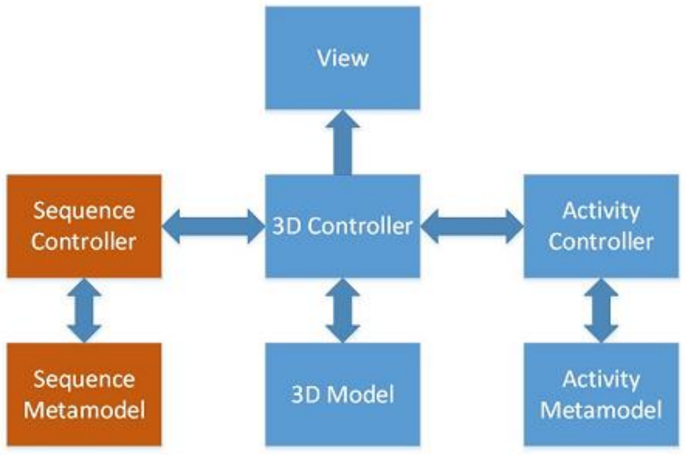


Figure 2. MMVCC Architecture.

3.3 Transformation of large desktop client to Web

When transforming a desktop application, it is important to determine the needs at first because there is a difference between transforming a simple form application and an advanced application using a lot of enhanced graphics. The backend is not discussed in this paper because there are many Server-Side JavaScript based languages (e.g. NodeJS), which perfectly can do everything that all main desktop languages like C++, C# or Java are capable of.

3.3.1 Transforming form applications

Transforming form applications is the easier of the two. There are mostly no problems encountered during the process as the current technologies for form-like applications on the Web are well developed and supported [13]. With HTML5 and CSS3 it is no more a problem. Using user interface Frameworks like Bootstrap the output display of the Web applications might be even better than the desktop client due to features like responsive design, high flexibility, ease of use and many reusable templates. Desktop form applications are using mainly 2D technologies, and 2D technologies are well developed and standardized for web therefore migrating 2D form applications from desktop to web is no longer a problem.

3.3.2 Transforming graphic applications

Graphic applications are much harder to transform and migrate than form applications. There is not yet a standardized way for migrating graphical desktop applications to web. In many cases transforming them to pure HTML and CSS is almost impossible. Although there are rare experiments, when even games were developed using advanced HTML5 and CSS3 hacks. These solutions tend to be ineffective and lagging [11]. This is due to current situation that 3D is not yet standardized for web and there are three main streams in developing such applications, these are HTML canvas, WebGL and CSS 3D transformations.

HTML5 canvas uses only 2D transformations and the 3D effects have to be self-done which leads to missing hardware optimized graphic rendering. WebGL is a JavaScript API for rendering interactive 2D/3D computer graphics. It is using HTML5 canvas and OpenGL ES as its base. Frameworks based on WebGL and WebGL itself is capable of creating advanced graphic with complex elements using hardware accelerated graphic rendering, but with using WebGL we lose the features of HTML features as DOM manipulation, input tags so it is important to choose WebGL only when needed. CSS 3D transformations are getting more popular and their functionality is being further expanded and in future it might be the universal way for nay graphic application. Now it has limitations on advanced graphic effects and complex geometrically shaped objects.

Difference between the WebGL approach and the pure CSS 3D transformations approach can be seen on Figure 3. There we see how the WebGL can be displayed together with HTML/CSS elements, but it itself does not contain these elements and cannot get use of the extended pros which these features offer. The simple graphic applications which does not require the use of complex three dimensional objects, shading and other advanced graphic features, as is the case of our prototype, can be effectively transformed into pure HTML/CSS and so get all the advantages of not needing any Web Graphical Engine. Problem with pure CSS 3D transformations is connecting elements in space, like arrows and lines. We were able to create such an arrow between elements in 3D space with only CSS 3D transformations, thus still maintaining all the pros which HTML and CSS brings us.

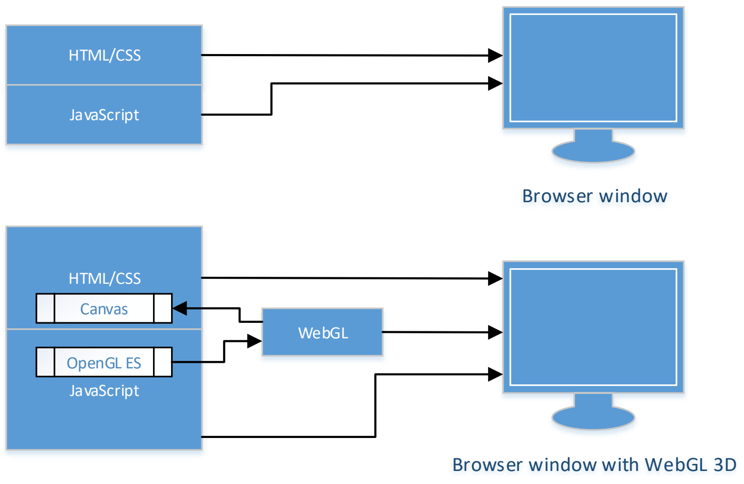


Figure 3. CSS3D and WebGL architecture for Web App.

4 Conclusion and future work

In migration of graphic desktop applications to web, there is not yet a standardized way of doing it. It is very important to well measure the system to be migrated and choose the appropriate technology. Although losing the power of pure HTML/CSS and encountering problems with synching HTML with WebGL, for applications with complicated geometries and a lot of visual effects is the best way to use WebGL engine for their web applications. Dealing with simple shapes and effects it is very good to stay with pure HTML and CSS 3D transformations. The boom wave of WebGL is on decline and there is a lot of work put into CSS 3D transformation libraries and tools and it looks like in future this could expand even more.

Transforming natural language and written text into diagrams is our nearest goal. We plan to extend our prototype for interpreting natural language into three dimensional diagrams. These diagrams we plan to generate automatically using Force-directed graph layout algorithms which allows weighting of the vertices and the edges for less complex and easier to comprehend 3D diagrams and models.

Acknowledgement: This contribution is the partial result of the project Research of methods for acquisition, analysis and personalized conveying of information and knowledge, ITMS 26240220039, co-funded by the ERDF.

References

[1] Koike H.: The role of another spatial dimension in software visualization. In: *ACM Transactions on Information Systems (TOIS)*, (1993)

[2] Koike H. and Chu H-CH.: How does 3-d visualization work in software engineering? In: *ICSE '98 Proceedings of the 20th international conference on Software engineering*, ICSE Proceedings, (1998)

[3] Irani, P., Ware, C.: Diagrams based on structural object perception. In: *AVI '00 Proceedings of the working conference on Advanced visual interfaces*. AVI Proceedings, ACM New York, (2000). pp. 61–67.

[4] Gil, J. and Kent, S.: Three dimensional software modelling. In *ICSE '98 Proceedings of the 20th international conference on Software engineering*, ICSE Proceedings, (1998).

[5] McIntosh, P.: X3D-UML: User-Centred Design. Implementation and Evaluation of 3D UML Using X3D. PhD. Thesis, RMIT University, (2009).

[6] Duske, K.: *A Graphical Editor for the GMF Mapping Model*. (2010). http://gef3d.blogspot.com/2010/01/graphical-editor-for-gmf-mapping-model.html

[7] Pilgrim J. and Duske K.: Gef3d: A framework for two-, two-and-a-half-, and three-dimensional graphical editors. In: *Proceedings of the 4th ACM Symposium on Software Visualization*, SoftVis '08, pages 95–104, New York, NY, USA, 2008. ACM.

[8] Biederman, I.: Recognition-by-components: A theory of human image understanding. Psychological Review, vol. 94, (1987). pp. 115–147.

[9] *OMG Unified Modeling Language (OMG UML)*, Infrastructure, Version 2.4.1, [Online], Available: http://www.omg.org/spec/UML/2.4.1/, Accessed: March 1, 2014

[10] Clark K.: *Creating 3D worlds with HTML and CSS*, http://www.keithclark.co.uk/articles/creating-3d-worlds-with-html-and-css/

[11] Gregorovic L., Polasek I., and Sobota B.: Software model creation with multidimensional UML. In: *International Conference on Research and Practical Issues of Enterprise Information Systems*, The 23rd IFIP World Computer Congress, Daejeon, Korea, (2015)

[12] Gregorovic L. and Polasek I.: Analysis and Design of Object-oriented Software using Multidimensional UML. In: *I-Know 2015. 15th International Conference on Knowledge Technologies and Data-driven Business*, Graz, Austria. ACM. (2015)

[13] Puder A.: Extending desktop applications to the web. In: *Proceedings of the 2004 international symposium on Information and communication technologies* (ISICT '04). Trinity College Dublin, pp. 8 - 13. (2013)

[14] Belan R.: Transformation of UML Combined Fragments from 2D to 3D. In: *Proceedings: of the 11th Student Research Conference in Informatics and Information Technologies Bratislava*, (2014). pp. 245 – 250.

The Seamless Wi-Fi Handover in SDN Architecture

Rastislav BENCEL*

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
rastislav.bencel@stuba.sk

Abstract. Nowadays mobile devices are present everywhere around us and the number of them is still increasing. Because of this fact the requirements as constant connectivity and mobility within network reach significant place in their research. In this article we focus on issues with the transition of mobile devices between access points called handover within Wi-Fi technology. The solution of this handover collaborates with SDN architecture which allows better management of functioning in networks. This article presents Wi-Fi technology, SDN architecture, existing solutions for performing handover and the proposal of new architecture for performing handover.

1 Introduction

Nowadays mobile devices e.g. notebooks, smartphones are everywhere around us and has an influence on the load of wireless networks. Traffic has been increased every year and from year 2000 to year 2013 it was 18 times greater [1]. Telecommunication operators have to invest big budget to upgrade their cellular networks. One option for solving this problem is using Wi-Fi technology for providing telecommunications services by Wi-Fi offloading. In this case, it is necessary to ensure a stable connectivity for mobile devices and handoff stations. Not only telecommunication operators are interested in Wi-Fi technology with providing stable connectivity, but also many big companies with large buildings want to have Internet services e.g. VoIP. Therefore it is needed to focus on handover process during moving end's user station in Wi-Fi technology.

SDN architecture represents a new trend in computer networks and it is important to explore also this area as a possibility of providing better solution to solve problem with performing handover. The paper is focused on handover processing and issues associated with it.

The paper is organized as follows. Section 2 contains a description and a limitation of performing handover of Wi-Fi technology, Section 3 and 4 contain a description of SDN architecture and existing solutions of performing handover. Section 5 contains a detailed description of a new centralized architecture for performing handover based on SDN architecture.

* Doctoral degree study programme in field: Applied Informatics
Supervisor: Assoc. Professor Margaréta Kotočová, Institute of Computer Systems and Networks, Faculty of Informatics and Information Technologies STU in Bratislava

2 Wi-Fi

Standard IEEE 802.11 [2] offers high bandwidth for clients within small area. In this case, if we want to cover bigger area, we have to use more access points to ensure sufficient quality of service. This type of wireless network is called WLAN Enterprise which covers a large area by the signal. Within this enterprise networks we can observe some problems associated with the mobility of a user from one AP to another AP. During this handover the user looses connectivity and all data during this transition. The standard in the first version is not created with the intention of providing handover and this is the reason that time needed for making handover is too long e.g. at least a few seconds. Later standards such as IEEE 802.11f, 802.11k ,802.11r introduce some more improvements for the time reduction. The main reasons having the greatest impact on handover time are:

- Discovery process – client’s device scans channels to find available Wi-Fi networks.
- Authentication process – client’s device has to do an authentication once again during reassociation of the process.

2.1 Discovery process

The discovery process has two modes which can be used for finding available Wi-Fi network:

- Active – a device sends a *probe request* on every channel and waits for the answer in the form of *beacon frame* from access points using the same channel, in which the request has been sent.
- Passive – the device only scans channels and expects *beacon frames* from APs which periodically send these *beacon frames*.

The discovery process can take long time until the particual device finds network where the connection can be done. As a result of this reason, IEEE introduces standard 802.11k which helps to reduce time of discovery process. A client can require from his current AP the list of possible Wi-Fi networks from the neighbourhood. This feature can but also can not reduce time of the discovery process. [2]

2.2 Authentication process

The process, which is present during every client reassociation, is the authentication process. Standard extensions 802.11f and 802.11r reduce the impact on the length of handover time. Standard 802.11f skips this authentication process and replaces it with exchanging AP context within the distribution system of network. Standard 802.11r also provides the authentication process but in the different form. The authentication process is done before receiving the *reassociation response frame* from new AP. In the basic standard it is performed after received response frame. This extension can offer handover about 50 milliseconds. Time interval under 50 ms is not detectable by the human ear which is requested within VoIP. Many vendors use their own solution for providing handover and do not use this standard. [2]

2.3 Limitation of technology

The concept of Wi-Fi technology is not proposed with the main goal of providing handover within network but with high throughput in network. This is the reason why the performing handover does not achieve the level of providing quality of services which is needed for traffic such as VoIP.

Wi-Fi technology uses a distribution view on network where every node of infrastructure can work alone. This view on network can have the impact on performance of the whole network because AP decision can have an improved performance on the local part of network but also decreasing performance on the other parts of network. This distribution view is not able to do an optimal decision due to not having all knowledge about network in devices. Handover decision is done by client side based on signal strength. [3]

3 Software-defined Networks

Software-defined networks represent a new trend in current network world [4]. Main features of this type of networks are decoupled control and data plane and centralized view on network. SDN architecture has three layers [5]:

- Application – represents a layer which provides services to network as applications which communicate with control layers via northbound API. This API is not standardized nowadays and depends on an implementation of SDN controller.
- Control – represents a control layer in which SDN controller is located. SDN controller detects all underlying devices via southbound API. This API uses standardized protocol and that is the reason for using devices from different vendors.
- Infrastructure – represents a layer in which all network devices are located. These devices forward only packets on the basis of rules which are setup by SDN controller via southbound API. [6]

3.1 Wireless networks within SDN

SDN for wireless network provides better options in view of management networks due to using centralized view. Better controlling over the wireless network helps to achieve better network performance. In wireless networks we can ensure controlling the following properties:

- radio resource allocation,
- monitoring and optimizing interferences in network,
- load balancing,
- performing handover.

4 Existing solutions

Solutions for solving problem with Wi-Fi handover are described in this section. These solutions are

- Personal access point – non SDN solution.
- Control connectivity – SDN solution.
- Odin framework – SDN solution.

4.1 Personal access point

This solution uses a creation of virtual access point [9]. Every client has its own virtual access point which moves across the network. A client does not observe any changes of the access point. This proposal must have a central component which decides about performing handover. This access point does not have the same role as in traditional Wi-Fi network and this is the reason why access point is called as wireless termination point (WTP). This name is also used in CAPWAP protocol. WTP does not perform all functions as AP in traditional network and represents only the termination point of wireless shared medium access. We recognize three types of WTP:

- Local MAC – all functionality is implemented on WTP.

- Split MAC – one part of functionality is implemented on WTP and the other is implemented on access control component. A part of functionality which is implemented on WTP is often critical in response time but also there can be some other type of functionality.
- Remote MAC – all functionality is implemented on access control component.

The solution shows new way how to access the performing handover in Wi-Fi networks.

4.2 Control connectivity

This solution concept [1] uses SDN architecture as a basis of network. All components including WTPs also called as terminal or base station support OpenFlow protocol for managing rules in flow tables by SDN controller. Wi-Fi part of network uses one SSID and one BSSID on all WTPs. Client's side does not recognize, on which physical WTP is connected. All decisions in network are done in SDN controller and handover is performed only by changing rules in flow tables. Flow of data changes from an old WTP to the new one. The control of the connectivity of the solution supports multichannel Wi-Fi environment and the change of channel is achieved by Channel Switch Announcement (CSA) message which is in IEEE 802.11 standard.

4.3 Odin framework

This solution uses SDN architecture too and it is focused on framework which provides programmability within WLAN enterprise. Situated services are on the top of this framework as applications. One of these apps is determined to provide handover process. The solution uses PAP and only one channel within Wi-Fi [7]. An extension of Odin [8] introduces multichannel environment and a decision on the basis of the load of WTPs and the signal strength.

5 Solution proposal

In this section we introduce our solution containing modified architecture of SDN which is able to provide seamless Wi-Fi handover without modification of the end user device. All changes touch only network infrastructure. Architecture is shown in Figure 1. Within the proposed architecture we introduce two new components HDS (Handover Decision Server) and AFCP (Additional Function of Control Plane) for which we define interfaces with other components of SDN.

This solution uses the concept of personal access points (PAP) where every client has its own associated virtual access point which moves with him across the whole network. Moving this PAP between access points (in our work represented as WTPs) is performed by the handover. WTPs use split MAC concept of PAP. We use our own communication protocol for ensuring performing handover, to report statistic information, and to exchange PAP between WTPs.

The decision of performing handover is moved from client's side (end user's mobile station) to HDS. More about HDS is noted in the separate section below. The handover is performed on the basis of the properties which are the load of WTPs, signal strength, data rate, delay, jitter and more properties can be added. Wi-Fi environment uses more channels to avoid interferences which occur during the usage of only one channel.

5.1 Proposed components of the architecture

According to mentioning some information above the components of the proposed architecture are:

- HDS (Handover Decision Server) – the main task of this component is the performing decisions about handovers on the basis of reports from WTPs. After this decision HDS controls this process which consists of parts such as moving PAP and informing AFCP for changing network data plane.

– AFCP (Additional Function of Control Plane) – function of this component adds a functionality to the control plane of SDN controller which controls forwarders in infrastructure layer. If the transition of PAP between WTPs occur then AFCP changes forwarding of client's traffic through SDN controller on the basis of the request of HDS and it is shown in Figure 2.

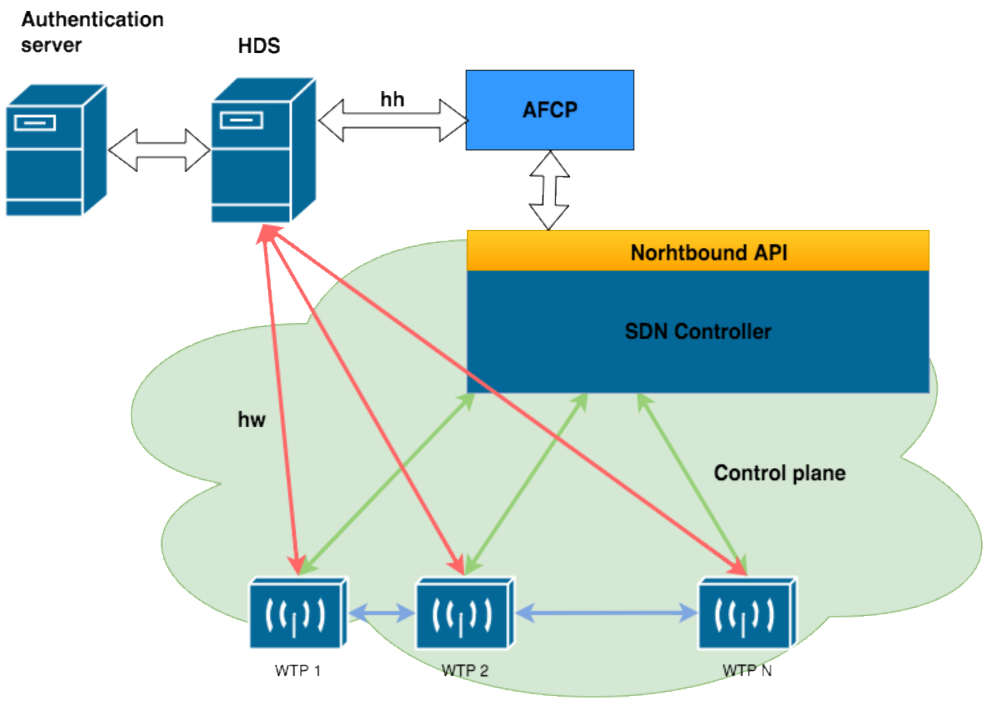


Figure 1. The proposal of architecture for providing seamless handover in SDN.

5.2 Architecture interfaces

The proposed architecture contains several interfaces. We offer two of them. These interfaces are shown in Figure 1 and are called *hw* and *hh*. Other interfaces are proposed by by other organizations. The solution is independent of the implementation of these interfaces.

Interface *hw* represents connection between HDS and WTPs. Data is transmitted in both directions via this interface. This data is needed for the successful handover. *Hw* includes data such as information for adding new WTP to network, creating and deleting PAP, reporting stats about state of network.

A function of the second interface *hh* unifies the control of data forwarding in network because northbound API for adding services, as applications to network, dependeds on an implementation of SDN controller. As the result of this dependency, we decide to create this interface to provide an implementation independence of HDS from SDN controller.

5.3 Handover process

Handover process shown on Figure 2 can be divided into following parts:

1. Reporting statistical information from WTPs to HDS via *hw* interface.
2. On the basis of reports from WTPs, HDS decides on the destiny of handover.

3. If HDS decides to perform handover, it will send the request to new WTP with information about the creation of PAP.
4. After HDS received successful response about creating PAP, it sends the request to AFCP for changing client's traffic flow to new WTP. In case that new WTP works on different channel, HDS also sends to old WTP message to inform client about changing channel by CSA message. Client's side thinks that only his WTP changes and does not know about performing handover.
5. After successful change of client's traffic flow, HDS sends the request to old WTP about deleting PAP from it.
6. Handover process is successfully finished.

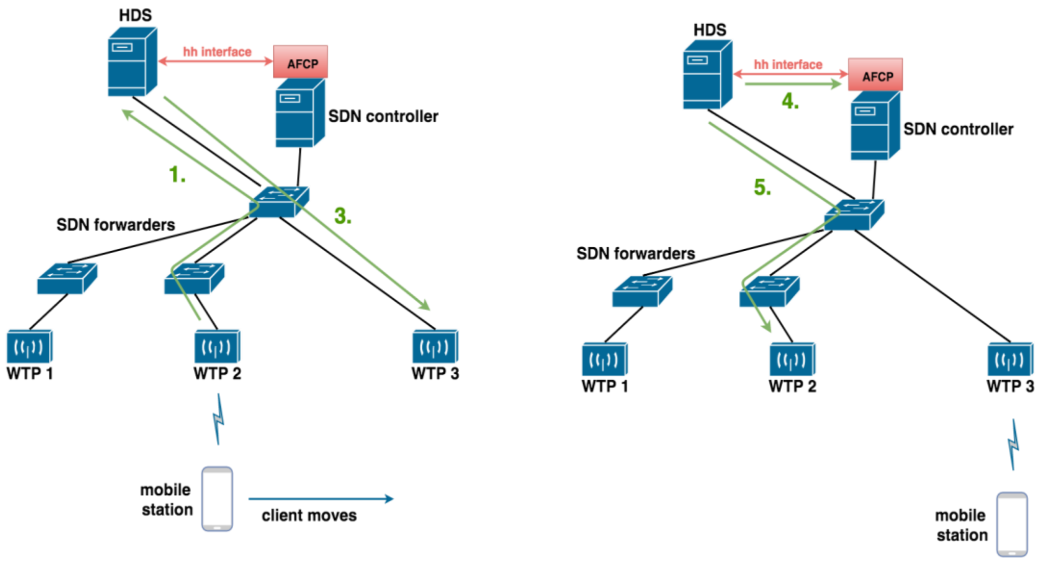


Figure 2. Example of handover process for mobile station.

5.4 Differences from other existing solutions

The advantage of this proposed architecture provides good management within handover process for multichannel Wi-Fi environment which is not directly present in Odin framework [7] but it can be found in its extension. Comparing with Odin framework our solution is focused on providing the mobility and Odin on programming resources within them Odin is possible to provide handover process and other services. Odin's extension can provide handover process for multichannel environment with the load and the signal strength of evaluation. In our solution we count with not only the load and the signal strength but also with parameters as data rate, delay, jitter. Our solution will optimize performance in the whole network by transition between WTPs. Comparing with solution for control connectivity [1] our solution uses the different approach to PAP.

6 Conclusion

The article contains analysis of Wi-Fi and SDN article in context of handover process. We propose new architecture based on SDN which provides handover process. We use PAP concept to solve problems with long discovery and authentication process. PAP should remove difficulties from previous processes. In our next work we will focus on implementation, evaluation and comparison of our solution with others.

Acknowledgement: This work was partially supported by the Scientific Grant Agency of Slovak Republic, grant No. VEGA 1/0616/14.

References

[1] Nakauchi, K.; Shoji, Y., "WiFi Network Virtualization to Control the Connectivity of a Target Service," in *Network and Service Management, IEEE Transactions on* , vol.12, no.2, pp.308-319, June 2015

[2] 802.11-2012, IEEE Standard for Information technology--Telecommunications and information exchange between systems Local and metropolitan area networks--Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," in IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007) , vol., no., pp.1-2793, March 29 2012

[3] SCHILLER, Jochen. 2003. Mobile Communications (2nd Edition). Pearson, 2 edition. 492p, ISBN 978-0321123817

[4] Fei Hu; Qi Hao; Ke Bao, "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation," in *Communications Surveys & Tutorials, IEEE* , vol.16, no.4, pp.2181-2206, Fourthquarter 2014

[5] Software-Defined Networking: The New Norm for Networks, *Open Networking Foundation*, 2012

[6] OpenFlow Switch Specification, Version 1.5.1, *Open Networking Foundation*, Mar. 2015

[7] Lalith Suresh; Julius Schulz-Zander; Ruben Merz; Anja Feldmann; Teresa Vazao, “Towards programmable enterprise WLANS with Odin” in *Proceedings of the first workshop on Hot topics in software defined networks (HotSDN '12),* ACM, pp. 115-120. 2012

[8] Rangisetti, A.K.; Baldaniya, H.B.; Kumar, B.P.; Tamma, B.R., "Load-aware hand-offs in software defined wireless LANs," in *Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2014 IEEE 10th International Conference on ,pp.685-690, 8-10 Oct. 2014

[9] ZAN, L., WANG, J. a BAO, L. Personal AP Protocol for Mobility Management in IEEE 802.11 Systems. In: *MobiQuitous 2005. The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*. 2005. s. 418–425.

Authenticating Users Based on How They Pick up Smartphones

Kamil BURDA[*]

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
`kamil.burda@stuba.sk`

Abstract. User identification and authentication methods become increasingly reliant on the use of biometrics – observing the user's physiological or behavioral characteristics – to reduce user inconvenience and increase security. On smartphones, one may leverage the potential of touch screen and motion sensors (accelerometer and gyroscope) to extract useful behavioral biometrics. This paper presents a novel approach to authenticate users based on the way users pick up a smartphone on a table or in their front pockets, an activity performed frequently every day, using the smartphone's accelerometer sensor.

1 Introduction

Mobile devices, especially smartphones, have become nearly ubiquitous, with people using them every day for numerous applications, such as messaging or social media. It is easier more than ever to compromise the security of smartphones and users' personal data therein given that users deliberately choose weaker authentication mechanisms for convenience. This is understandable, as traditional mechanisms, such as text-based passwords, are difficult and error-prone to type on small touch screens. Lock patterns, a popular choice for a phone unlock mechanism due to the simplicity of drawing patterns, can be deduced by tracking the corresponding smudges on the touch screen.

One way to enhance the security of devices without disrupting the user experience is to track user's biometric characteristics. Physiological biometrics determine who the user is, such as user's fingerprints, irises or face. Behavioral biometrics observe the user's behavior, such as typing behavior (also called keystroke dynamics) or movements (hand gestures, walking). While behavioral biometrics are considerably less accurate compared to physiological, they do not require any additional hardware, are more difficult to replicate than physiological biometrics and are more resistant against differences in the environment (such as the amount of light in the surroundings or background noise) [14]. Behavioral biometrics are therefore generally recommended to be used as an additional security mechanism or a part of a multi-modal biometric system. Another use of biometrics is to adapt content and user interface of applications for each user individually.

[*] Doctoral study programme in field: Software Engineering
Supervisor: Assoc. Professor Daniela Chudá, Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies STU in Bratislava

In order for a system (an application on a device) to identify or authenticate users based on biometrics, the system needs to observe the user and create a user model. The system first requires the user to perform activities sufficiently long enough so that the system is then able to train itself to recognize the user. The trained data is called a user template. Once trained, the system then logs user's activities from chosen devices or sensors (such as accelerometer for movement-based biometrics). The logged data are processed to output useful features (such as mean acceleration or velocity for movements). The values of extracted features are then matched against the user template (using statistical methods or classifiers). Successfully matching (classifying) the user may result in successful user identification or authentication, depending on the goal of the system. For user authentication, the system determines whether the extracted features match the user's template. User identification is a more difficult task, as the system must distinguish each stored template in the system to match the extracted features.

The accuracy of a biometric system is usually evaluated by the False Acceptance Rate (FAR) and False Rejection Rate (FRR). FRR is the probability that the system fails to recognize the authorized user. FAR is the probability that the system incorrectly recognizes an unauthorized user as an authorized user. FAR and FRR can be adjusted according to a threshold in the specific classifier used in the system. If a threshold is set to a value such that FAR equals FRR, it is called the Equal Error Rate (EER).

The user identification and authentication accuracy for smartphone-based behavioral biometrics can still be increased by improving existing methods (as the research of behavioral biometrics for smartphones is still relatively new) and by discovering new reliable biometrics that can be combined with other biometrics in multi-modal biometric systems.

The behavioral biometrics for smartphones have so far focused primarily on gestures and taps performed on a touch screen and detecting body movements such as walking or simple hand gestures using the accelerometer sensor. One of the body movements that has not been thoroughly examined is the movement of picking up the smartphone on a surface or the user's pocket. Users perform this movement frequently every day, and thus has a great potential as another method of authenticating them seamlessly.

In this paper we determine whether the movement of picking up a phone can serve as another viable biometric for user authentication. We aim to verify this assumption based on an experiment in which 30 participants were tasked to pick up a smartphone from a table upon receiving a fake notification, dismissing the notification and putting the phone back on the table. The experiment was performed with standing and sitting posture and placing the phone on a table and in one of user's front pants pockets, giving four sets of experimental data for each participant.

This rest of this paper is structured as follows. Related work is discussed in Section 2. The proposed method is described in Section 3. Section 4 describes the proposed evaluation of the method. The last Section concludes the work.

2 Related work

Many studies focused on recognizing users based on their gait (walking patterns) using the smartphone's accelerometer [1, 7, 8]. In [8], participants were asked to walk on a flat surface (carpet) up and down a hallway. Before features could be extracted from the accelerometer data, segments related to walking were manually filtered. Features, including x, y and z direction of acceleration and the magnitude of acceleration, were evaluated for each walk segment separately. A similar study [1], also gathering walk data and splitting them into segments, uses raw readings from the accelerometer for x, y and z values and applies wavelet transform on them to gather useful features in time and frequency domains. The study also considers different walking patterns, such as walking in circles or abruptly changing walk direction. These studies ignored data not associated with any of the gait-based activities, including the movements of picking up and placing the phone to a pocket.

Study [9] observed how different positions of a phone affect authentication accuracy for walking users. The results showed that the position of the phone in a hand and in a pocket exhibit different characteristics and thus may have a significant impact on user authentication accuracy. On the other hand, holding the phone on left or right hand did not affect user authentication accuracy, likewise for having the phone in user's left or right pocket.

Study [3] aims to authenticate users based on the movement performed when answering or placing a phone call, using the smartphone's accelerometer and orientation sensor. The proposed system starts tracking the user once the user presses a button to answer/place a call and brings the phone to the ear. As acknowledged by the authors, this biometric measure can authenticate users transparently as does not require them to perform additional actions in order to train the system. While the accelerometer and the orientation sensor are used to track user's movement, it is unclear from the study how the system recognizes the moment the user brings the phone to the ear (to determine the end of the movement). The proximity sensor, commonly found in smartphones, could be used as an indicator that the user is holding the phone to the ear.

The movement of picking up and placing the phone was also subject to research [4–6, 13]. In general, accelerometer is used to gather raw data, along with gyroscope and magnetometer in several studies. In [4], participants were asked to pick up the phone from a table in two different postures – sitting and standing. In [5], the gathered data were split into three phases – picking up the phone from a table, holding the phone to the ear and placing the phone back on the table.

While the movement trajectories were already studied in [4], it is assumed that the start and end of these movements (picking up the phone, dismissing the notification, placing the phone) exhibit unique movement patterns for each user.

Compared to previous studies, this paper evaluates the movement in multiple postures – sitting and standing – and in different places of the phone – a table and user's front pants pocket. Additionally, the method takes the application context taken into account to determine the start of the movement, making the method applicable in real-world scenarios.

3 Method

This Section describes the proposed method to authenticate users based on the movement of picking up a smartphone from a table and one of user's pockets.

Users receive notifications on a daily basis - SMS messages, calendar reminders, e-mails or phone calls. From the moment user picks up the phone, the application logs data from the accelerometer sensor until the user explicitly dismisses the notification.

Using the sensors should be kept to a minimum to avoid excessive resource usage (especially on mobile devices), including battery power. Therefore, it is strongly recommended that the sensor data be logged only from the time to pick up the phone until the user dismisses the notification (reads a message, postpones a reminder, accepts the phone call).

3.1 Features

The following proposed features are extracted from the raw accelerometer readings:

- minimum, maximum, mean and standard deviation of acceleration for x-, y-, z-acceleration and magnitude of acceleration (computed as $m = \sqrt{x^2 + y^2 + z^2}$), respectively,
- pick-up time – time from receiving a notification until its dismissal.

The features discussed are computed for each pick-up movement separately, starting with the moment a notification is generated and ending when the accelerometer readings show that the phone is no longer being moved, or a specified number of seconds after the dismissal if the phone is still being moved. It is yet to be determined how many seconds in the latter situation is optimal for each examined posture.

The pick-up time may not always be a reliable feature in real-world scenarios. For example, the length of a received message may delay the user dismissing the notification. For the controlled experiment described later in this paper, the displayed messages are short enough to not delay the user for too long.

3.2 Matching users

Once computed, the features form a vector of values to act as an input to a classifier. In this paper, we choose the *k*-Nearest Neighbors (*k*-NN) algorithm and the Support Vector Machine (SVM) due to their good classification performance. SVM as a binary classifier is convenient for user authentication as the features for the phone owner can be treated as one class of data and features for non-owners (impostors) as another class.

In practice, however, it is unusual for a smartphone to be used by more than one person. A biometric system therefore has no plausible way to gather enough data from non-owners. Given this limitation, one-class SVM [12] is used as another classifier to determine whether the extracted features closely match the phone owner's.

Given that accelerometer readings can be represented as time series, the readings can be compared with each other in terms of their similarity to identify or authenticate users. For this purpose, the Dynamic Time Warping (DTW) algorithm is used as another user matching method to compare against *k*-NN and SVM.

4 Evaluation

This Section describes the evaluation of proposed method in the Section 3. The method is verified based on a controlled pilot experiment.

4.1 Experiment description

30 users in total participated in the experiment composing of four simple tasks. In the first task, the participants were instructed to sit down on a chair in front of a table. The participant first places a phone on the table. After a small period of time, the phone vibrates and generates a notification (an SMS message or a calendar reminder with sample text), the participant picks up the phone, reads and dismisses the notification and places the phone back on the table.

Each task is performed four times in total in order to generate a sufficient amount of sample data for each user without unnecessarily prolonging the experiment. In real-world scenarios, the number of samples should be much higher (at least in tens) to provide good accuracy of classifiers used for this method.

In the subsequent tasks, the participant is instructed to stand up and repeat the task, then repeat the same task while sitting and placing the phone in the front pants pocket and finally while standing and placing the phone in the front pants pocket. Figure 1 shows a sample of raw accelerometer readings for a pick-up movement from a table while a user is sitting. The first two valleys represent the phone vibration as a result of a generated notification. The subsequent readings show the user picking up the phone, reading the notification, dismissing the notification and putting the phone back on the table.

It is apparent that phone vibrations are undesirable and that this segment must be filtered prior to extracting features. Simply removing the segment containing phone vibration is not a viable solution because the user may pick up the phone while the phone is vibrating. It is yet to be determined whether phone vibrations can be safely filtered in the frequency domain without accidentally filtering the useful data. As an alternative, if it is known which vibration pattern corresponds to which notification, we may be able to build different user templates for different types of notifications without filtering the vibration patterns. Each user performed the experiment with the same phone, *HTC Desire* with Android operating system (version 2.2). Apart from the accelerometer sensor,

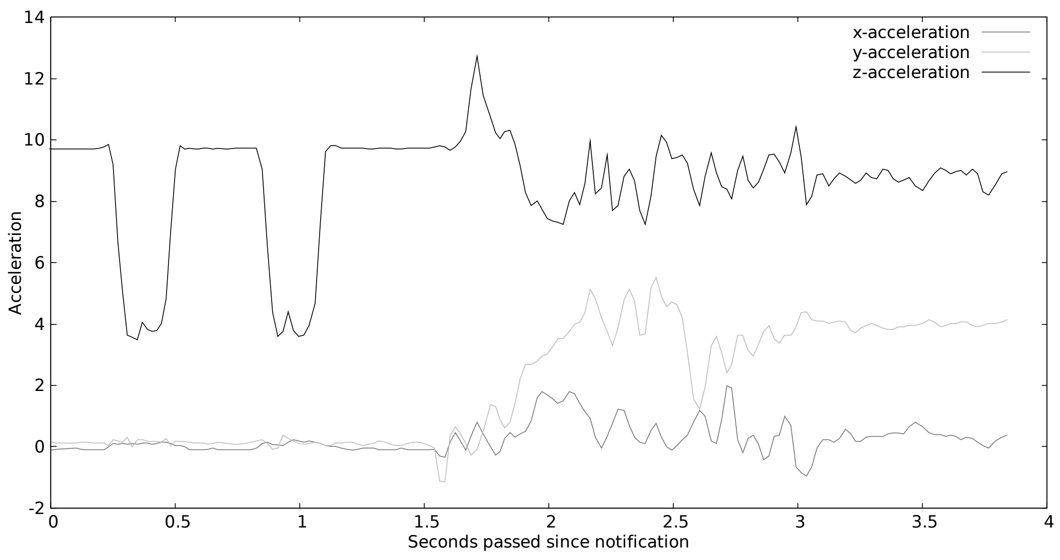


Figure 1. Raw accelerometer readings for a pick-up movement from a table for the sitting posture.

other sensors were recorded as well, including the magnetometer and touch screen, reserved for further evaluation of the method in the future. The data were logged continuously during the entire experiment for each user individually. Each action (start of task, notification generated, notification dismissed) was associated with a time stamp to help identify segments belonging to the pick-up movements. The labeled segments also help identify each task with different postures and phone positions to treat each task separately during the evaluation.

There are several real-world factors not considered in the experiment that could affect users' movement patterns, such as table height, distance of the phone from the user, different pocket size or users having no pockets at all. Users may react differently given the application context, such as receiving a message, a reminder or a phone call.

Users may pick up the phone with either the left or right hand, causing the trajectories to differ. Given the conclusions in [9] for walking movements, it is expected that picking up or placing the phone with the left or right hand will not impact user authentication accuracy.

4.2 Evaluation of results

To evaluate the performance of the proposed method for one user, we set the user as the phone owner and other users as non-owners. The classifier then determines the false positive rate and false negative rate, considered to be FAR and FRR, respectively. Once the performance is evaluated for each user, we compute the mean FAR and FRR for all values of FAR and FRR from each user, which finally represents the accuracy of the proposed method for user authentication.

It is expected that the extracted features will differ for different postures (standing, sitting) and different phone positions (table, pocket) for the same user. The authentication performance is therefore evaluated for the following combinations:

– postures and positions individually,

– all postures combined,

– all positions combined,

– all postures and positions combined.

Likewise, the performance of classifiers specified in Section 3.2 is compared.

5 Conclusions and future work

This paper proposes a method to authenticate users based on how they pick up smartphones from a flat surface (a table) or their front pants pockets. The goal of the pilot experiment is to determine whether the movement of picking up a smartphone can be used to distinguish users and can thus be used as another viable behavioral biometric characteristic. Additionally, the contribution of this paper is to determine how different postures and phone positions affect the user authentication accuracy for this type of movement.

The future work comprises actual evaluation, including using the DTW algorithm on accelerometer readings and gathering features from additional data logged during the experiment - touch screen data and magnetometer readings. To match the pick-up time closer to the real-world scenarios, it may be redefined as the time the user picks up the phone until the phone stops moving - that is, until the phone's accelerometer readings become stable for a specific time period.

Beside the pick-up movement, placing a smartphone back on the table or users' pockets will also be examined as it may exhibit unique biometric characteristics. In this case, however, we cannot precisely determine when the user stops placing a phone as the user performs no explicit action afterwards. Given the accelerometer sensor, we may be able to detect if the phone stops moving as per the redefinition of the pick-up time feature.

Given that in the pilot experiment we also recorded additional sensors, such as magnetometer and touch screen, we are able to compute additional features that may improve the performance of the method, such as velocity-related features, phone's orientation or pressure applied on touch screen (which proved to vastly improve the accuracy of user authentication in related works [2, 10, 11]).

The movement of picking up a phone can further be divided to three segments – start of motion (grabbing the phone), motion proper (drawing the phone closer to user's body) and end of motion (the phone stops moving) – as each of these segments exhibits vastly different values, especially in terms of phone's acceleration.

Acknowledgement: This work was partially supported by the Scientific Grant Agency of Slovak Republic, grant No. VG 1/0774/16.

References

[1] Boyle, M., Klausner, A., Starobinski, D., Trachtenberg, A., Wu, H.: Gait-based User Classification Using Phone Sensors. In: *Proceedings of Conference on Mobile Systems, Applications and Services*.

[2] Chang, T.Y., Tsai, C.J., Lin, J.H.: A graphical-based password keystroke dynamic authentication system for touch screen handheld mobile devices, vol. 85, no. 5, pp. 1157–1165.

[3] Conti, M., Zachia-Zlatea, I., Crispo, B.: Mind How You Answer Me!: Transparently Authenticating the User of a Smartphone when Answering or Placing a Call. In: *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*. ASIACCS '11, ACM, pp. 249–259.

[4] Feng, T., Zhao, X., Shi, W.: Investigating Mobile Device Picking-up motion as a novel biometric modality. In: *2013 IEEE Sixth International Conference on Biometrics: Theory, Applications and Systems (BTAS)*, pp. 1–6.

[5] Kunnathu, N.: Biometric User Authentication on Smartphone Accelerometer Sensor Data.

[6] Makaram, Y.: The Phoney Lift: Using Accelerometers to Identify People.

[7] Nickel, C., Derawi, M., Bours, P., Busch, C.: Scenario test of accelerometer-based biometric gait recognition. In: *2011 Third International Workshop on Security and Communication Networks (IWSCN)*, pp. 15–21.

[8] Nickel, C., Wirtl, T., Busch, C.: Authentication of Smartphone Users Based on the Way They Walk Using k-NN Algorithm. In: *2012 Eighth International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP)*, pp. 16–20.

[9] Primo, A., Phoha, V., Kumar, R., Serwadda, A.: Context-Aware Active Authentication Using Smartphone Accelerometer Measurements. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 98–105.

[10] Saevanee, H., Bhatarakosol, P.: User Authentication Using Combination of Behavioral Biometrics over the Touchpad Acting Like Touch Screen of Mobile Device. In: *International Conference on Computer and Electrical Engineering, 2008. ICCEE 2008*, pp. 82–86.

[11] Saevanee, H., Bhattarakosol, P.: Authenticating User Using Keystroke Dynamics and Finger Pressure. In: *6th IEEE Consumer Communications and Networking Conference, 2009. CCNC 2009*, pp. 1–2.

[12] Schölkopf, B., Williamson, R.C., Smola, A.J., Shawe-Taylor, J., Platt, J.C.: Support Vector Method for Novelty Detection, vol. 12, pp. 582–588.

[13] Shrestha, B., Mohamed, M., Borg, A., Saxena, N., Tamrakar, S.: Curbing mobile malware based on user-transparent hand movements. In: *2015 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 221–229.

[14] Zheng, N., Bai, K., Huang, H., Wang, H.: You Are How You Touch: User Verification on Smartphones via Tapping Behaviors. In: *2014 IEEE 22nd International Conference on Network Protocols (ICNP)*, pp. 221–232.

Scalable Personalized Recommender System

Adam LIESKOVSKÝ*

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
adamliesko@gmail.com

Abstract. Requirements and priorities for modern recommender systems grow and change hand in hand with the continuous growth of Internet users. In this paper we focus on aspects such as scalability, flexibility and speed of recommender systems, which nowadays play an important role. We propose novel hybrid recommender system, which utilizes user behaviour and context, items' content and items' popularity and recency. We evaluate our approach in the domain of news articles, processing streaming data and computing recommendations online, in real-time. Conducted evaluations showed the benefits of using article trendiness as a strong component in the process of computing recommendations.

1 Introduction and related work

Information overload is nowadays a serious issue on Web. Users don't have the time, neither the resources or skills to manually search and filter for items which they seek. Personalized recommendations are great way to reduce and eliminate this problem. Furthermore, it helps users to discover new items, which they wouldn't be able to find by the means of standard navigation. Other than that, personalized recommendations help with increasing the commercial revenue as well. Therefore, it is a fairly popular topic not only in the academic world, but also between the tech giants like LinkedIn [1], Netflix [2] or Amazon [3], where recommendations play a key role in the user experience.

Another prevailing problem, which appears with the enormous traffic and number of users the popular sites have to serve, is the aspect of scalability. Recently, methods like distributed systems, GPU matrix computations or client-server computation distribution [4] have enabled for real-time processing of streaming data on a large scale.

Majority of established approaches for personalized recommendations fall into the categories of collaborative filtering and content based recommendations. Each of these approaches comes with it's downsides (e.g., cold start problem of a new item or new user, long tail problem), which hybrid recommender systems usually help to eliminate [5]. For long, data processing and computations at large scale have been limited by existing technology or software. With the introduction of map-reduce computation model, now also available through in-memory

* Master degree study programme in field: Software Engineering
Supervisor: Dr. Michal Kompan, Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies STU in Bratislava

computation, or the massive growth of NoSQL databases, recommender systems are nowadays able to achieve the required scalability and efficiency needed for the Web scale.

Nowadays, collaborative filtering approaches, for example using a scalable k-NN algorithm [6] or matrix factorization [7, 8], present the state of the art method for recommender systems. Most successful and widely used are matrix factorization processing user-item association matrix [9].

In [9] Shi et al. present a comprehensive survey of the state-of-the-art collaborative filtering and latest challenges in the domain of recommender systems. They identified the incorporation of social recommendations and additional user interaction data (e.g., context), cross domain and group collaborative filtering as the arising key challenges to the future of collaborative filtering.

2 Scalable and flexible recommendation approach

In this paper, we propose a scalable hybrid recommender system, with focus on a fast response time, scalability and flexibility in regard to the system workload and available resources. Abstractly, it can be separated into two subparts, first consisting of a real-time online computation of recommendations, the second of computing offline user models. Our hybrid approach can be classified as a meta-level, mixed or weighted, depending on the form of final recommendation aggregation. In the following sections, we describe respective recommender modules for hybrid recommender system in the domain of news articles, as outlined on the Figure 1.

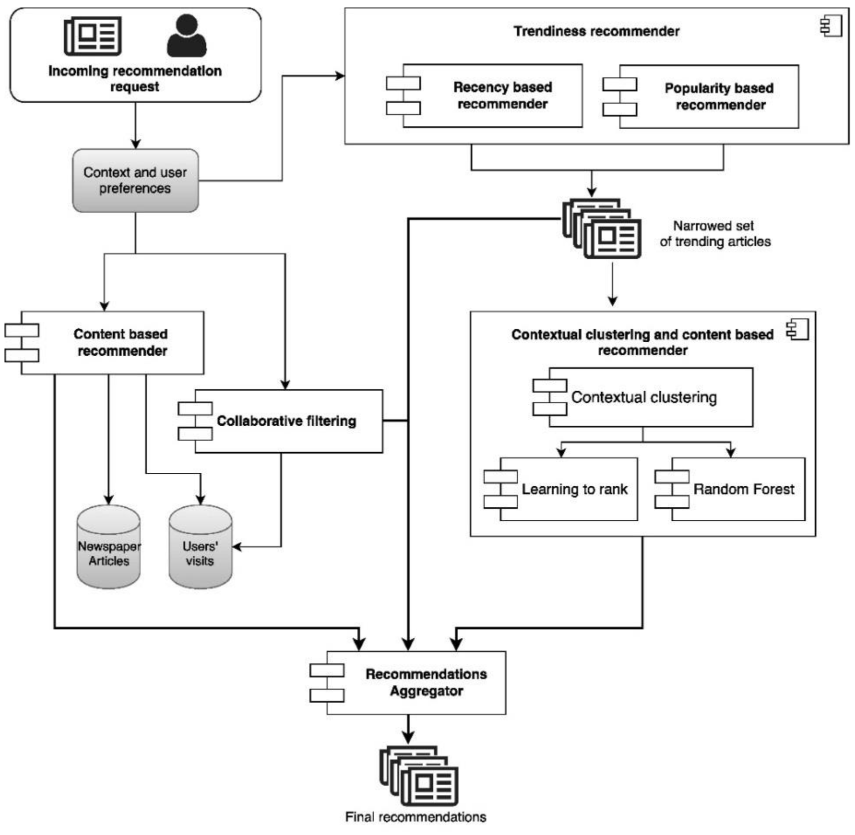

Incoming recommendation request
Trendiness recommender
Recency based recommender
Popularity based recommender
Context and user preferences
Narrowed set of trending articles
Content based recommender
Collaborative filtering
Contextual clustering and content based recommender
Contextual clustering
Learning to rank
Random Forest
Newspaper Articles
Users' visits
Recommendations Aggregator
Final recommendations

Figure 1. Proposed scalable hybrid recommender approach.

2.1 Content based recommendation

Research in natural language processing (NLP) and its approaches to content similarity estimation using *LDA* (*Latent Dirichlet allocation*) [10] or *LSA* (*Latent semantic analysis*) [11] have recently become fairly popular and successful. If combined with SVD (Singular value decomposition), these algorithms provide a scalable way to cluster items and infer topics based on their content. In our work we opted for a simpler and faster approach, utilizing direct content similarity estimation between item pairs.

On top of static item content comparison, we incorporate sentiment and relevance of extracted keywords or entities into the recommendation process. This information can be extracted from the content of items, e.g., in the domain of news articles we use article title and text as source of keyword extraction. This way, we can identify user's probable attitude towards certain topics (election parties, war etc.), entities (e.g., Steve Jobs, England) or extracted keywords. Given this information, we can narrow the recommendations to the users with items containing similar sentiment or attitude in general towards certain entities or topics, in which the user has previously shown interest.

2.2 Collaborative filtering

Since Netflix prize [8], matrix factorization has become a state-of-the-art method for computing recommendations with collaborative filtering. We chose alternating least squares (*ALS*) algorithm, which can compute latent factors describing users and items from user-item association matrix. ALS algorithm provides us with an option to specify a weight to a user-item interaction, if it turns out that it is desirable to differentiate between user clicking on a certain recommendation or a user naturally visiting an item. More specifically we are using *ALS-WR* approach, as described in [12]. *ALS-WR* uses a normalization parameter $lambda$, which tackles scalability and sparsity issues in matrix factorization. One of the nice properties of *ALS-WR* is that the accuracy of the method monotonically improves with the number of algorithm iterations.

ALS model is computed periodically in the background with latest subset of user-item interactions narrowed by their occurrence time. Computation of a model is distributable and we can easily set number of iterations based on the current system workload. The periods between model re-computation can be also adjusted according to the workload, which also adds to the flexibility property of our proposed approach.

2.3 Popularity and recency of items

Popularity and recency are two aspects of items, that we combined together to form a so-called trendiness recommender. We model popular and recently created or updated items in certain time intervals. In addition to global counters, we monitor the popularity and recency of articles in the subsets of items, as per their categorization by content. For popularity, we use integer counters of item visits occurrences in the respective time interval. For recency aspect, we propose a more sophisticated approach. Our goal is to be able to assign the same value of recency to an article published 10 minutes ago, as to an article published 1 hour ago (example thresholds). We are able to achieve it, by using Equation 1[1], which creates a gauss like figure of recency in respect to the times of item creation and their last update. $Decay$ and $scale$ parameters enables the fine grained time decay property of recency recommender module.

$$recency(article) = w_1 * timePenalty(\text{published}_{at}) - w_2 * timePenalty(updated_{at}) \qquad (1)$$

$$timePenalty(time) = exp(-\frac{max(0,\ |time - \ published_{at}| - penaltyStart)^2}{2(-\frac{scale^2}{2} * \log(decay))}$$

[1] https://www.elastic.co/guide/en/elasticsearch/guide/current/decay-functions.html

We either use the subsets of our so-called trending articles directly as top-n recommendations, or push them further down our method, where they serve as an input to some of the more computational expensive methods for recommendation generation.

Our reasoning behind this approach is strongly tied with the current trends of navigation on the Web. On majority of sites you can find lists of most popular items, new and current items, that narrows the information and navigation space for users. This especially applies to the domain of newspaper articles, where users are visiting news portals in order to find new and current information. These items could stand a higher chance of actually being visited by user, because on average, majority of users tend to be interested in some of the most popular and recent topics. We proved this assumption correct in our evaluations.

2.4 Contextual recommendations and clustering

Nowadays, context is getting ubiquitous and as seen in other works [13], it is a great source of information, which we can use for clustering user recommendation requests. In our method, we use contextual data like geo-location, gender, estimated age or salary, ISP and similar properties. By using streaming *k-means* algorithm, specifically the *kmeans++* implementation [14], we are able to perform clustering in real-time, while maintaining up-to-date representation of the clusters. Each recommendation request is assigned to a cluster, based on the properties of the request context. Inside of each of these cluster, we hold machine learning models used for classification and ranking items of items. Classification outcome and rank of items depend on the relevance and confidence of respective items being clicked or visited by user.

For classifying and ranking items we use two scalable and parallelizable approaches: decision trees (*Random forest*) and learning to rank (*Listnet*). Learning to rank and *Listnet* algorithm specifically, have already been proved [15, 16] to be a suitable choice for large scale machine learning model construction. In our approach we are using them as estimators of user's probability of actually clicking on a recommendation. Input to machine learning models is a narrowed subset of trending articles and outcome is a set of top-n recommendations with respective confidence weights, based on the learned model for a specific cluster, to which a recommendation request was assigned depending on contextual data. These models are updated and reconstructed periodically in the background with the latest data.

2.5 Aggregation of recommendations

Our proposed approach provides us with variety of possible combinations and aggregation of recommender modules within our hybrid system. The aggregation process is exactly the place, where the flexibility and adaptive properties of our approach can show off. Under ideal conditions and state of a recommender system (low workload), our approach uses all of the described recommender modules. Trendiness module acts as a meta-level recommender, which generates input subsets for other recommender modules (e.g., contextual). Collaborative filtering and content based recommendations, which work with constructed user models provides recommendations based upon the user's behaviour and user models. Recommendations from the respective recommender are then grouped and their confidence is summed up, along with the weight of each of recommender modules.

Our approach eliminates cold start issue of a new user in the system. Even for an unknown or new user, we still have on-line contextual data and information about the currently visited item. Altogether with popularity and recency information, we can identify and recommend relevant articles to the user. Incoming request is assigned to a cluster by the on-line information and a machine learning model (learning to rank, random forest) ranks subset of trending articles according to the predicted cluster and user. As a response to the recommendation request, we present top-n recommendations to the user.

3 Evaluation

As explained in the previous section, one of the key points of our proposed approach is the focus on popularity and recentness of articles. In order to prove this hypothesis, we evaluated the effect of adding popularity and recency aspects to a simple k-nn recommender. In the k-nn algorithm, nearest neighbors were chosen by the co-occurrence factor based on item to item similarity [3]. Similarities were computed using log likelihood ratio of impression occurrences. Evaluations were performed on an offline dataset, consisting of ~12 million impressions gathered from slovak online newspaper publisher SME.SK during a week in October 2009.

As shown in the Table 1, we were able to improve precision@10 and NDCG@10 metrics by over 100%, when we added popularity and recency aspects into the recommendations process. With items impressions, we have only binary scale (seen/not seen) which is unable to fully utilize NDCG properties. This effect can be seen also by the similar improvement rate of both of the used metrics.

Table 1. Evaluation of article trendiness increase in precision and NDCG metrics.

No. of nn	Without popularity and recency		With popularity and recency (exp. boosting)			
	precision @10	NDCG@10	precision @10		NDCG@10	
2	0.0086	0.0410	0.0179	**+ 108%**	0.0803	**+ 96%**
3	0.0093	0.0411	0.0190	**+ 104%**	0.0792	**+ 93%**
5	0.0114	0.0495	0.0222	**+ 95%**	0.0906	**+ 83%**
7	0.0104	0.0453	0.0228	**+ 19%**	0.0913	**+ 102%**
10	0.0108	0.0446	0.0243	**+ 125%**	0.0952	**+ 113%**
15	0.0107	0.0414	0.0229	**+ 114%**	0.0860	**+ 108%**
20	0.0109	0.0418	0.0229	**+ 110%**	0.0840	**+ 101%**
30	0.0103	0.0412	0.0225	**+ 118%**	0.0809	**+ 96%**

4 Conclusions and future work

In this paper, we have proposed hybrid recommender system with focus on the scalable and extensible architecture, flexibility of the system and ability to adapt to the system workload. Generated recommendations are computed with the use of up-to-date information, whether it is users behaviour and context or item content. As a unique feature of our approach, we highlight recommendation requests clustering based on the user’s context, which reduces information space and enables for frequent and continuous updating of underlying machine learning models used for generating recommendations.

So far, we have performed only a basic and partial evaluation of our proposed hybrid recommender approach. We yet have to evaluate usage of clustering of incoming requests, scalability and flexibility of our approach. We plan to perform online evaluations with streaming data, where we will focus on CTR metric. Furthermore, we will perform rigorous and repeatable evaluations concerning scalability and accuracy of our proposed recommender system with prepared datasets created from acquired streaming data.

In terms of extending our proposed method, there are several opportunities for improvement and experimenting. Inclusion of information gathered from social data could serve as a great addition to the user model. For example, if we had performed latent topic analysis (LDA/LSA), we could map these identified topics to not only topics identified inside the items content, but also to the user’s interests, as expressed on their social network profiles.

Acknowledgment: This work was partially supported by the Scientific Grant Agency of Slovak Republic, grant No. VG 1/0774/16.

References

[1] R. Sumbaly, J. Kreps, and S. Shah, “The ‘Big Data’ Ecosystem at LinkedIn,” *Int. Conf. Manag. Data (SIGMOD 2013)*, pp. 1–10, 2013.

[2] X. Amatriain, “Big & Personal: data and models behind Netflix recommendations,” *Proc. 2nd Int. Work. Big Data*, pp. 1–6, 2013.

[3] G. Linden, B. Smith, and J. York, “Amazon.com Recommendations Item-to-Item Collaborative Filtering,” *IEEE Internet Computing.*, vol. 7, no. 1, pp. 76–80, 2003.

[4] A. Boutet, D. Frey, and A. Kermarrec, “HyRec: Leveraging Browsers for Scalable Recommenders Categories and Subject Descriptors,” pp. 85–96, 2014.

[5] V. Vekariya and G. R. Kulkarni, “Hybrid recommender systems: Survey and experiments,” *2012 2nd Int. Conf. Digit. Inf. Commun. Technol. its Appl. DICTAP 2012*, pp. 469–473, 2012.

[6] X. Yang, Z. Zhang, and K. Wang, “Scalable collaborative filtering using incremental update and local link prediction,” *Proc. 21st ACM Int. Conf. Inf. Knowl. Manag. - CIKM ’12*, p. 2371, 2012.

[7] G. Takács, I. Pilászy, B. Németh, and D. Tikk, “Scalable Collaborative Filtering Approaches for Large Recommender Systems,” *J. Mach. Learn. Res.*, vol. 10, no. 6 , pp. 623–656, 2009.

[8] R. M. Bell and Y. Koren, “Lessons from the Netflix prize challenge,” *ACM SIGKDD Explor. Newsl.*, vol. 9, no. 2, p. 75, 2007.

[9] Y. U. Shi, M. Larson, and A. Hanjalic, “Collaborative Filtering beyond the User-Item Matrix: A Survey of the State of the Art and Future Challenges,” *ACM Comput. Surv.*, vol. 47, no. 1, pp. 1–45, 2014.

[10] B. Chen, “fLDA : Matrix Factorization through Latent Dirichlet Allocation,” *New York*, pp. 91–100, 2010.

[11] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, *Recommender Systems Handbook*. Boston, MA: Springer US, 2011.

[12] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan, “Algorithmic Aspects in Information and Management,” Proc. of the 4th Int. Conf., AAIM 2008, Shanghai, China,R. Fleischer and J. Xu, Eds. Berlin, Springer Berlin Heidelberg, 2008, pp. 337–348.

[13] C. Palmisano, A. Tuzhilin, and M. Gorgoglione, “Using context to improve predictive modeling of customers in personalization applications,” *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 11, pp. 1535–1549, 2008.

[14] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, “Scalable K-Means ++,” *Proc. VLDB Endow.*, vol. 5, no. 7, pp. 622–633, 2012.

[15] S. Shukla, M. Lease, and A. Tewari, “Parallelizing ListNet training using spark,” *Proc. 35th Int. ACM SIGIR Conf. Res. Dev. Inf. Retr. - SIGIR ’12*, p. 1127, 2012.

[16] T. Niek, “Scaling Learning to Rank to Big Data Using MapReduce to Parallelise Learning to Rank,” University of Twente, Twente, Netherlands, 2014.

Towards Rule Based Refactoring

Lukáš MARKOVIČ*

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
`lukass.markovic@gmail.com`

Abstract. This paper presents the advanced approach to source code refactoring using XML technologies and rule based expert system. The article also describes particular refactoring problems such as dependencies between code smells, or order effectiveness of refactoring operations. Our attention is dedicated not only to well-known code smells, but also similar concept called anti-patterns. Some of these problems are used as example to explain possibilities of expert system with defined rules in refactoring process. Main part of the article describes a proposal of software system, able to perform automated refactoring using rule based decision making based on code smells dependencies analysis.

1 Introduction

Refactoring, first defined by Martin Fowler, is a process of changing a software system source code in a way, that it does not affect external behaviour, but yet improves its internal structure [2]. Besides refactoring definition, Fowler also describes the target of refactoring – code smells, as indications of a deeper problem in source code structure. In addition to code smells, there are also anti-patters, firstly mentioned by Andrew Koenig [3]. They are described as commonly occurring solutions that can cause problematic consequences [1]. There is only small difference between code smells and anti-pattern. These terms are therefore often being confused, or used without a difference. In fact, by definition, only code smell should be the real target of refactoring, because anti-patterns can also cause system problems. Removing an anti-pattern can therefore cause change in system behaviour, and that is not consistent with the refactoring definition. Despite of these terminology problems, refactoring process is nowadays used for removing both code smells and anti-patterns from the source code. Software system thus becomes easier to maintain, alternate, adapt and further develop.

However refactoring costs time, it is very necessary process during the life cycle of almost every software system. Therefore, there is a great effort in refactoring automatization. Also here in Faculty of Informatics and Information technologies was created a lot of research studies in field of refactoring automatization [7][8], [9][11]. This paper, which is related to previous research works proposing the rule-based refactoring [9], [11], is dedicated to the less known approach to

* Master degree study programme in field: Software Engineering
Supervisor: Dr. Ivan Polášek, Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies STU in Bratislava

refactoring automatization using XML technologies in combination with expert system based decision making, for obtaining the best possible result from automated refactoring process.

2 Automatized source code refactoring

Refactoring automatization is a very difficult process that varies based on the selected technologies of representing the source code, searching for anti-patterns or code-smells and refactoring of these problems itself.

Code smell search and refactoring can be performed on many source code representations. Technique of code smell search and refactoring itself is subsequently unwound based on selected representation. Table 1 describes some of the possible source code representations and corresponding techniques of code smell search and refactoring.

Table 1. Possible representations and corresponding techniques of code smell search and refactoring.

Source code representation	*Search technique*	*Refactoring technique*
plain text	metrics	text processing tools
AST	AST libraries, metrics	AST libraries
JSON	JavaScript, …	JavaScript, …
XML	XPath, XQuery	XSLT, XQuery, XQuery Update

As presented, automated refactoring can be performed on a clean code itself. Then, many well-known source code metrics can be used for code smell search [10], such as a *lines of code*. On the other hand, refactoring is very hard to perform on such a "*clean*" representation. One option is to use text processing tools, that are able to work with simple text, but this option is complicated and unnecessary.

Abstract syntax tree is probably the most widely used method for refactoring automatization. This method is based on a tree representation that removes all insignificant source code elements, for example parentheses or spaces. Many languages, or IDE's provides libraries, that are able to programmatically work with AST representation. Typical example is Eclipse AST library that is used in many refactoring and source code manipulation tools under Eclipse [4].

Despite that most widely used technique for refactoring automatization is abstract syntax tree, in our work we focused on XML technologies. XML, similarly as JSON notation allows serialization of source code. Afterwards, searching and refactoring in such representation can be simply performed with technologies that are able to manipulate with given representation [5].

3 XML based refactoring process

XML language, which is a widespread language for data serialization can be also used for a source code representation. There are several tools able to convert source code into XML representation and vice versa[1]. Many XML technologies can be then used for search of an anti-patterns and code smells [6]. There is also a lot of useful Technologies [6], such as XSLT[2] or XQuery[3], for refactoring itself.

One problem with XML representation is, that every source code file is represented as a single XML document. When refactoring is performed, there is often a needs to work with more than one file in time. For example, refactoring method "*push up method"* brings selected method

[1] http://www.srcml.org/
[2] https://www.w3.org/TR/xslt-30/
[3] https://www.w3.org/TR/xquery-3/

into other class, that is often in its own file. This means, there is need to work with system source code as with single document, which removes the need to solve documents interconnections.

Solution to these problems are XML databases, that are kind of NoSQL documents databases. Project can be represented as a single collection of documents in this kind of databases. Each query into project collection works with project de facto as one document, and this effectively solves presented problem.

Nowadays, there is only few actively supported native XML databases. These includes:

- BaseX
- eXist
- MarkLogic
- Sedna

Despite that XML databases are relative old, not a wide spread technology, they are still supported and they have actively developed an interface for many languages, such as Java – XQJ API[4].

XML databases find only small usage nowadays, but are highly suitable for application in XML based refactoring process. They also usually offer processor of many XML technologies, such as XPath, XSLT or XQuery. XML Technologies can be used to search and mark a concrete problem and also to refactor problems in the source code, when using XML databases. XML based processes of anti-pattern or code smell search and refactoring will be described in the following subchapters.

3.1 Code smells search using XML technologies

As described in [7], XPath language is very suitable tool for XML based code smells search. Despite its simple, not hard to learn language, it is able to effectively locate and return desired nodes from XML documents. Using simple XPath expressions, it is possible to detect a lot of common, as well as little-known code smells. However, there are some reasons to select different XML associated language for code smell search.

XQuery, which used XPath for node locations, is a language for querying in XML documents. Native XML database systems are often using XQuery as main language to query into database. According to this and also needs to store information’s about found problems, what XPath is not able to, the XQuery language is probably best technology for search and also refactor code smells, as described in subchapter 3.2.

3.2 Refactoring of code smells using XML technologies

Refactoring itself is the most difficult part of refactoring process. There are several possibilities to select a refactoring technology, for example XSLT [7]. Another options are XQuery and its extension XQuery Update. These are able of functional programming over XML documents, and are more sufficient in case of proposed tool.

Refactoring, as an operation of source code structure changing, requires modifying of XML documents in the database. Only XQuery itself is insufficient for this purpose, because it does not contain operations of modifying the document itself. This problem is solved by *XQuery Update* extension. Example of XQuery script is provided in fragment code below. Very simple refactoring operation *Remove exception throw* is shown, in which *delete* operation is XQuery Update command.

```
declare variable $tag external := "CR1";
       for $node in xquery:eval(fn:concat("//", $tag))
return
       delete node //throw
```

[4] http://xqj.net/

4 Rule based refactoring

Refactoring has many areas, which need to be considered during the process. One of them is the influencing of the code smells each other's.

Influencing means that by removing one code smell, other code smell can arise on a given place. Typical example is, that by removing *middle man, message chains* code smell often arises. Besides this, positive code smell influencing can also be taken into mind. This means, that by removing one code smell on given place, other code smell can be removed as well. This behaviour is very interesting, knowing that every refactoring affects the structure of the code. In general, the target of refactoring is to remove code smells in effective way. This means with as minimum as possible refactoring operations. Removing code smell, which removes also other code smell on given position is better, than firstly removing inner code smell and then removing outer code smell.

These two kind of influences between code smells are simple to take into consideration when refactoring is performed in a manual process. But during refactoring automatization, it is hard to consider these influences. One possibility is to introduce rule based expert system, able to decide about optimal or sub optimal refactoring sequence. An expert system, which contains knowledge of code smell influences, can be then included into automated refactoring system. An expert system can select refactoring operations and their order based on search analysis of source code. Nowadays, there are few tools able to create such an expert system. Jess[5] tool, which represents expert system programing language based on Java language is one possibility which was also used in proposed tool.

4.1 Dependencies between code smells

Identification of positive and negative influences between code smells is necessary in order to build such a rule based on refactoring system.

Identification of influences between each code smell and anti-patterns is very difficult, because more than one hundred source code problems are identified. In proposed tool we focused on identification of influences between the well-known 22 code smells defined by Martin Fowler. Despite of that, the number of problems is reduced into 22, there can still be identified a lot of relations between them. The reason for this is that each code smell can be removed by application not only one refactoring operation. Each operation can arise, or remove different type of code smell.

Due to these complicated relations, it is necessary to introduce the concept of code smell groups, which can be presented in form of super classes for concrete code smells. Consequently, it is possible to simplify relations between code smells using these subclasses. Identified group of code smells are presented in Table 2 below.

Table 2. Classification of code smells into groups.

Group	*Code smells*
Bad Size	Large Class, Long Method, Long Parameter List
Bad Location	Feature Envy, Comments, Duplicated Code, Divergent Changes, Shotgun Surgery, Switch Statement
Bad Class Content	Data Class, Lazy Class, Feature Envy, Large Class
Bad Inheritance	Alternative Classes with Different Interfaces, Parallel Inheritance Hierarchies, Refused Bequest
Needless Part	Comments, Duplicated Code, Refused Bequest, Speculative Generality
Attribute Problem	Data Clumps, Temporary Field, Primitive Obsession
Bad Communication	Inappropriate Intimacy, Middle Man, Message Chains

[5] http://www.jessrules.com/jess/index.shtml

Given the complex character of some code smells, it is appropriate, to include them into more than one group. With such hierarchy of code smells, there is possibility to create complex views at both kinds of code smells relations. UML class diagram is a sufficient tool for creating such views, as can be seen on Figure 1, that presents positive dependencies between code smells.

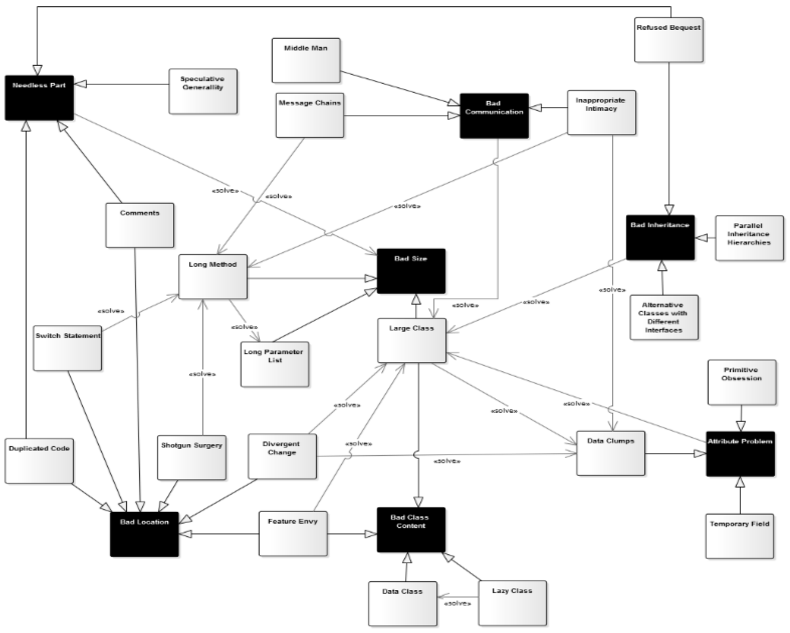


Figure 1. Positive dependencies between code smells.

Solid arrow in combination with <<*solve*>> stereotype is used for positive dependencies, as can be seen in Figure 1. On the other hand, for the negative dependencies (refactoring can cause another smell) <<*cause*>> stereotype and dashed arrow can be used and similar diagram can be created.

4.2 Design of refactoring rules

It is possible to straightforward design refactoring rules based on previous analysis of code smells dependencies. Based on given rules, rule engine can decide about selected refactoring operations.

Such refactoring rule can be very simple, in form of *if – then* rule, as presented below.

```
(defrule empty-catch-clause
              "Empty catch clause refactoring"
              ?o <-(JessInput {refCode == "ECC"})
              =>(add (new JessOutput ?o.code "LE")))
```

However, it can also take into consideration other conditions, for example the size of concrete problem, or, as presented, dependency to other found problem on given place. On the code fragment below, there is a simple Jess rule, which takes such dependency between code smells into account.

```
(defrule long-parameter-list
       "Long parameter refactoring with possible collision check"
       ?o <-(JessInput {refCode == "LPL"})
       (JessInput (parents ?parentsList))
       (not(test (?parentsList contains "LM")))
       => (add (new JessOutput ?o.code "IPO")))
```

Based on the second presented rule, there is only selected refactoring operation *Introduce parameter object* for refactoring of code smell *Long parameter list*, only if code smell *Long method* was not found in place. Otherwise, *Long parameter list* refactoring will not be executed, before *Long method* refactoring. On the other hand, based on first presented rule, *Log Exception* refactoring method is selected unconditionally as solution for every *Empty Catch Clause* problem.

5 Conclusions

This paper presented new, advanced approach to refactoring. Rule based refactoring, in combination with XML technologies, offers great potential into the future. XML database, that has strong processing power is a very important part of XML based refactoring process. In future work, there is a need to examine dependencies between all known code smells more precisely. Refactoring rules and strong refactoring tool can be created based on such analysis.

Acknowledgement: This contribution is the partial result of the project Research of methods for acquisition, analysis and personalized conveying of information and knowledge, ITMS 26240220039, co-funded by the ERDF.

References

[1] Brown, H.B., Malveau, R.C., McCormick, H.W, Mowbray T.J.: *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis.* John Wiley & Sons, Inc., (1998).

[2] Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D.: *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, (1999).

[3] Koenig, A.: Patterns and Antipatterns. In: *Journal of Object-Oriented Programming*, (1995), vol. 8, pp. 46-48.

[4] Kuhn, T., Thomann, O.: *Abstract Syntax Tree*. [Online; accessed November 20, 2006]. Available at: http://www.eclipse.org/articles/Article-JavaCodeManipulation_AST

[5] Mamas, E., Kontogiannis, K.: Towards Portable Source Code Representations Using XML. In: *WCRE '00 Proceedings of the Seventh Working Conference on Reverse Engineering*, IEEE Computer Society, Washington, (2000), pp. 172–182.

[6] Markovič. L.: *Refactoring support using transformations between Java a XML*. Bachelor's thesis, Slovak University of Technology in Bratislava, (2014).

[7] Markovič. L.: Refactoring Support Using XSLT Transformations. In.: *IIT.SRC 2014 Proceedings in Informatics and Information Technologies Student Research Conference*, Slovak University of Technology in Bratislava, (2014), pp. 457-462.

[8] Pipík, R., Polášek, I.: Semi-automatic refactoring to aspect-oriented platform. In: *CINTI 2013: proceedings of the 14th IEEE International Symposium on Computational Intelligence and Informatics*, Budapest, (2013), pp. 141-145.

[9] Polášek, I., Snopko, S. Kapustík, I.: Automatic identification of the anti-patterns using the rule-based approach. In: *SISY 2012: IEEE 10th Jubilee International Symposium on Intelligent Systems and Informatics*, Subotica (2012), pp. 283-286.

[10] Simon, F., Steinbrückner, F., Lewerentz, C.: Metrics based refactoring. In: *CSMR '01 Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, IEEE Computer Society, Lisbon, (2001), pp. 30–38.

[11] Štolc, M., Polášek, I.: A Visual based Framework for the Model Refactoring Techniques. In: *SAMI 2010, 8th IEEE International Symposium on Applied Machine Intelligence and Informatics*, Herľany, (2010), pp. 77-82.

Interactive System for Creation of Notes

Martin NEMČEK[*]

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
xnemcekm@stuba.sk

Abstract. We are overwhelmed by information from various topics. The challenge in education is to create notes which covers important subset of information. There are known methods to extract information from text. In this article we propose a system to extract the notes from text which are important for educational purpose, so it should create personalized notes for students. We use mainly syntactic text analysis. Notes are created by help of part-of-speech tags and dependencies between words in sentences. The outcome will be an interactive system for creating notes based on learned rules from user.

1 Introduction

Computers are not able to understand information in natural language. In our proposed system the notes are created from sentences by extracting relevant information from them. We use syntactic analysis of sentences and extract relations and dependencies between words from these sentences. The final result of our proposed method are personalized notes. The user will be able to modify the automatically created notes. The system will then learn new rules for sentence to note transformation from these changes and takes them into account for the next time.

2 Our proposed system

A rule consists mainly from two parts – *list of data of original sentence* and *list of data of note*. Each entry in *list of data of original sentence* and *list of data of note* contains these parts: relation name and list of grouped dependencies with the same relation name. Each dependency contains a governor token, a dependent token and its position considering all dependencies. The governor and dependent token consists of Part-Of-Speech (POS) tag and index of word in sentence to which is the token connected. Index of the token is bounded with a position of its word in sentence.

Dependencies from the second list are applied to sentence to create a new note. The rule may order to create a compound note from a sentence. The compound note is composited of some simple sentences. The positions in sentence on which the note should be split into smaller sentences are kept within the rule.

[*] Bachelor study programme in field: Informatics
Supervisor: Miroslav Blšták, Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies STU in Bratislava

When processing a sentence an applicable rule has to be looked up in database before creation of the note. Dependencies of rule and dependencies of the sentence being processed has to correspond to each other. Evaluation is based upon two conditions. The sentence that is being processed has to have the exact amount of entries in list of data of original sentence while these entries contain exactly the same relation names as the rule's relation names.

The applicable rule is found if these two conditions are met. However, the conditions can cause a situation that more than one rule is found. In this case we have to calculate the match probability of this sentence and the original sentence obtained from the rule. The rule with the highest probability of the match is applied.

Calculating the match consists of several steps. First, the POS tags match of governor and dependent tokens is calculated separately. Indices of governor and dependent tokens are calculated also separately. These first steps determine if the sentence contains arbitrary dependency with same value of POS tag or index. In followed step is determined a half-match of dependencies. Half-match of dependency is match of POS tag and index at the same time at governor or dependent token of dependency. We calculate matches of POS tags and index of governor or dependent token for every dependency. Finally, in the last step we calculate the number of absolute-matched dependencies. Absolute-match dependencies is the total match of POS tags and indices in governor and dependent tokens. Every step has assigned a rating. If a condition in the step is evaluated as true, the rating of the step is added to the final result. The final result is a percentage value of the match. The rating is based on importance of the step in calculating a precise match, while depending on the number of steps and dependencies, so the final result cannot exceed a limit of 100%. A pseudo code for an algorithm calculating the match is outlined in Algorithm 1 and specific example is shown in Figure 1.

Algorithm 1 Calculating match.

```
1: procedure CalculateMatch($sentence, originalDependencies$)
2:     $oneCompareTypeRating \leftarrow$ calculate percentage rating of one comparison
3:     for all $originalDependencies$ do
4:         if count($sentence, dependency$) = count($originalDependencies, dependency$) then
5:             $match \leftarrow match + oneCompareTypeRating$
6:         $counter \leftarrow counter$ + count($originalDependencies, dependency$)
7:     $oneCompareTypeRating \leftarrow oneCompareType/counter$
8:     for all $originalDependency$ do
9:         for all $dependency$ do
10:            for all $comparison$ do
11:                if applyComparison($sentence, comparison, dependency$) then
12:                    $match \leftarrow match + oneCompareTypeRating$
        return $match$
```

If rule look up does not find any applicable rule, it means that the system have not processed the same or similar sentence yet. A manual rules of parser are used in this case. The output of the parser is a note. A new rule is created based on the note. Dependencies of original sentences are taken and used to create a *list of data of original sentence*. This list is then assigned to the rule. Dependencies of note are used to create a *list of data of note* which is then also assigned to the rule. The sentence ends are determined depending on how many sentences the note contains. POS tags and indices of tokens are stated by the corresponding words of the original sentence and the newly created note.

By the principle of rule look up, the sentence being processed has to contain dependencies from the *list of data of original sentence* and also dependencies from the *list of data of note*.

The process of applying a rule has several steps. For each dependency in the list of data of original sentence, the respective dependency is looked up in sentence that is being processed. The word corresponding with dependent token from the looked up dependency is taken and added to the note on its index position. In case of dependency relation *nominal subject* the word corresponding with governor is also added. After processing all dependencies the last minor changes are done such as capitalization of the first letter of the note, splitting note into more sentences if rule defined so. Algorithm 2 shows pseudo code of the process of applying rule on sentence.

Algorithm 2 Applying rule.

```
1: procedure ApplyRule($sentence, rule$)
2:     $note$ ← new Note
3:     for all $ruleDependencies$ do
4:         $dependency$ ← findDependency($sentence$, $ruleDependency$)
5:         if isFound($dependency$) then
6:             add($note$, getDependent($dependency$))
7:             if isNominalSubject(relation($dependency$)) then
8:                 add($note$, getGovernor($dependency$))
9:     splitToSentences($note$, sentencesEnds($rule$))
       return $note$
```

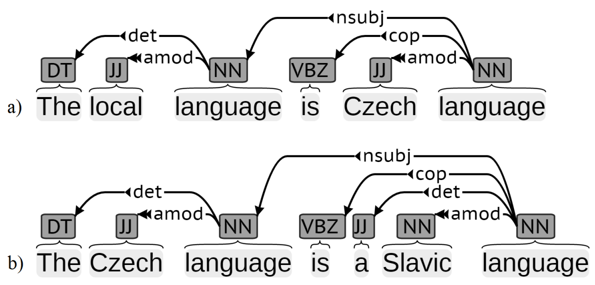

a)
DT
JJ
NN
VBZ
JJ
NN
det
amod
nsubj
cop
amod
The local language is Czech language
b)
DT
JJ
NN
VBZ
JJ
NN
NN
det
amod
nsubj
cop
det
amod
The Czech language is a Slavic language

Figure 1. Example sentences.

Let us consider example from Figure 1. We have rules for two sentences and we are processing the first one. In this situation, there are at least two rules which are applicable for the sentence *a*. Assume that we are calculating match with the sentence *b*. We iterate over all dependencies of processed sentence *a*. The first dependency is *det* with the governor token NN (noun) and index 3 and the dependent token DT (determiner) and index 1. First, we find out, if the sentence *b* contains any dependency with tokens NN or DT and index equals to 1 or 3. This is the separate calculation of POS tags and indices. Then, we try to find in the sentence *b* any dependency, which has dependent or governor token tag of type NN and index equal to 3 or tag of type DT and index equal to 1. This is only the half-match step. As the last step, we check if sentence b contains dependency, where the governor token is NN and index is equal to 3 and the dependent token is DT and its index is 1. If any of these step were matched, the rating of that particular step is added to the final result and iteration continues with following dependency until all dependencies were iterated over.

Acknowledgement: This work was partially supported by the Scientific Grant Agency of Slovak Republic, grant No. VG 1/0646/15.

PerfectPlaggie: Source Code Plagiarising Tool

Juraj PETRÍK*

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
petrik@fiit.stuba.sk

Abstract. This paper presents new tool for creating plagiarism copies of existing source codes, which supports multiple levels of plagiarism. The tool is especially helpful for creating large datasets of source codes, which will be used for benchmarking different methods for measuring source code similarity. The hypothesis of creation of such a tool is supported by perfect plagiarism experiment – which shows, that it is pretty easy to trick existing source code plagiarism detection systems.

1 Introduction

According to Merriam-Webster dictionary is plagiarism defined as:

– to steal and pass off (the ideas or words of another) as one's own
– use (another's production) without crediting the source
– to commit literary theft
– present as new and original an idea or product derived from an existing source

Plagiarism is serious problem in academic field – in 2002 a survey was performed, where students of Swinburne and Monash University were asked if they were ever engaged in academic dishonesty – 85.4% of Swinburne University students and 69.3% of Monash University students admitted it. [1]

Another study done at Faculty of Informatics and Information Technologies STU in Bratislava states that 33% of students admitted that they have ever created plagiarism during their study and 63% of them have ever given their work to someone to plagiarize it. As you can see software plagiarism is a big problem in academic sphere, but in commercial is too. [4]

For instance, events such as Google vs Oracle shows that software plagiarism is even worse problem than we thought. Jury did not find Google guilty of violating any Oracle's patents, but it was clear, that Google had plagiarized parts of Oracle's source codes. [5]

Software plagiarism is commonly detected by automated tools, but the problem with these tools is, that they are mostly not designed to resist sophisticated obfuscation attacks. [7]

* Doctoral degree study programme in field: Software Engineering
Supervisor Assoc. Professor Daniela Chudá, Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies STU in Bratislava

Another essential problem in this area is lack of large enough datasets with obfuscated source codes with different types of obfuscation. These datasets are crucial for benchmarking new methods and tools to fight plagiarism.

Therefore, idea of tool which will be able to automatically produce this dataset is described in this paper.

2 Related work

We can divide obfuscation attacks to two main categories [6]:

- Lexical changes
- Structural changes

Lexical changes can be done in text editor and do not require any special knowledge of programming language. Examples of such changes are adding/removing/modification of comments, source code formatting, changing identifier names.

Structural changes need some sort of special knowledge of programming language (understanding of the language) and are very language dependent. Loop changes (while<->for<->do while), condition changes (if<->case<->ternary operator) or statement order replacement are examples of structural changes.

2.1 Source code obfuscators

2.1.1 ARTIFICE

This tool performs transformations directly on source codes. Obfuscation types are as follows – renaming of variables, if else statements are transformed to ternary operators and vice versa, while statements are transformed to while and vice versa, expanding variable definitions, variable assignments.

2.1.2 ProGuard

ProGuard is a free Java class file shrinker, optimizer, obfuscator and preverifier. It detects and removes unused classes, fields, methods and attributes. It optimizes bytecode and removes unused instructions. It renames the remaining classes, fields, and methods using short meaningless names. [8]

It is able to create more compact code, make software harder to reverse-engineer or detect dead code. However, ProGuard works on byte code level – so there is need of additional steps to be done – compiling and decompiling of the bytecode to get obfuscated source code.

2.1.3 yGuard

yGuard is a free Java bytecode obfuscator and shrinker that improves your software deployment by prohibiting unwanted access to your source code and drastically shrinking the processed Jar files at the same time. [12]

This tool could do name obfuscation – replacing package, class, method, field names with random strings. This tools can also do code shrinking – code shrinking engine detects which parts of codes could not be reached from a set of given entry points. These not needed parts are then removed.

Similar to ProGuard this tools works on byte code level, so compiling to bytecode and decompiling from bytecode is required to get source code.

3 Perfect plagiarism experiment

To support the hypothesis, an experiment of manual plagiarism creation was done. The aim of this experiment was to show that it is possible to create “perfect” plagiarism source code – the modified version of original file will not be marked as suspicious (similarity percentage is below threshold value) by any of chosen source code similarity checkers.

For this experiment MOSS [3][10], JPlag [9] and SIM were selected as representants of plagiarism detection systems – because these tools are commonly used in academic area for evaluating student’s exams and are de facto used as standard benchmarking tools. Additionally, Simian was chosen, to see if there is any difference between special plagiarism detection systems and system used for software refactoring.

3.1 Source code sample

Red-black tree java implementation downloaded from the internet website providing source code samples was used in this experiment. This data structure implementation was chosen because it is typical student programming assignment at universities. Java as programming language was selected because it is most used programming language worldwide, also it is the most popular language among students.

3.2 Plagiarising

The goal of this experiment is too show, that it is possible to create perfect plagiarism in relatively short time. The person who was doing these obfuscations to achieve perfect plagiarism is simulating multiple levels of programming skills – to divide these copies by difficulty. Additionally, to simulate automatic obfuscation, the programmer was doing this changes without knowledge, what is this programming actually doing. In next three chapters these levels with results are described – first means that only beginner level is required to do this changes, second level requires advanced knowledge of language and third required semi-expert skill in this language.

3.2.1 First level

Just basic changes were done to the source code:

- Code formatting
- Comments removal
- Renaming of classes, methods and variables

Overall length of these changes was 10 minutes.

3.2.2 Second level

First level changes were done, plus some advanced changes:

- Loop changes (for->while, while->for…)
- Added new constants
- Negation of conditions
- Variable types
- Line ordering

Overall length of these changes was 30 minutes.

3.2.3 Third level

First level changes and second level changes were done, plus some more advanced changes:

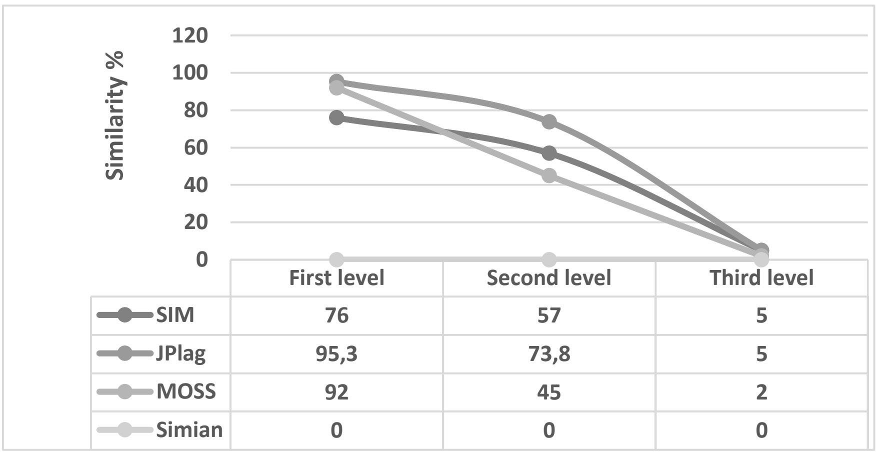
- Parameters order
- Variable modifiers
- Wrapping of return values
- Merging of some methods
- Splitting of some methods

Overall length of these changes was 30 minutes.

You can see example of this obfuscation level in Figure 2 (left side is original source code and on right side is obfuscated copy).

3.3 Discussion

Figure 1. Results of experiment represents results of completed experiment. First and second level obfuscations are not problem for plagiarism detection systems, however for Simian even first level changes are real problem. But third level changes are big deal even for specialized plagiarism detection tools – mainly it is because of excessive amount of unnecessary code added (wrapping).



	First level	Second level	Third level
SIM	76	57	5
JPlag	95,3	73,8	5
MOSS	92	45	2
Simian	0	0	0

Figure 1. Results of experiment.

When we are adding a lot of unneeded code, these tools are unable to detect this kind of situation, so the similarity percentage is naturally declining.

As we can see, it is easy to confuse standard plagiarism detection tools and it only took about one hour – it is surely faster than to do assignment on your own. Another alarming finding is that to do these changes it is not needed to be an expert in programming language, even the programmer does not need to know what the program is doing in real. This problem is not presented only in tested tools, but in other tools too. [11]

Based on these results I realized, that it is possible to create automated tool for creating obfuscated source codes. These obfuscated source codes (produced by the tool) will not be detected as suspicious (similarity value will be bellow threshold value) – to create perfect plagiarism by machine.

```
for (;;) {
 if (x.compareTo(current.element) < 0)
  current = current.left;
 else if (x.compareTo(current.element) > 0)
  current = current.right;
 else if (current != nullNode)
  return current.element;
 else
  return null;
}
```

```
while (true) {
 if (x.compareTo(getCurrentNode().getNodeElement()) > ZERO) {
  setCurrentNode(getCurrentNode().getRightNode());
 } else if (x.compareTo(getCurrentNode().getNodeElement()) < ZERO) {
  setCurrentNode(getCurrentNode().getLeftNode());
 } else if (getCurrentNode() != getNillLeaf()) {
  return getCurrentNode().getNodeElement();
 } else {
  return null;
 }
}
```

Figure 2. Source code obfuscation example.

4 PerfectPlaggie

Name of this tool is derived from “Perfect plagiarism” and plagiarism detection system Plaggie [2]. In next few chapters proposed design of this tool and planned supported obfuscation types are described.

4.1 Design

Figure 3 displays important architectural parts of PerfectPlaggie tool. These parts are GitHub crowler, file picker, obfuscator, tester.

GitHub crowler part will be searching for suitable projects on GitHub and downloading them. Suitable project means, that the project will have unit and integration tests with enough code coverage. Also it will ensure, that the downloaded project will be unique.

File picker part will pick only files, that will have 100% integration and unit test coverage – this is crucial for tester part of the PerfectPlaggie. It will also select files, that are interesting for dataset creation. The selection will be based on metrics such as LOC, NOM, MCC etc.

Obfuscator part will obfuscate source code files from file picker part. There will be multiple options of obfuscations – they are described in next chapter.

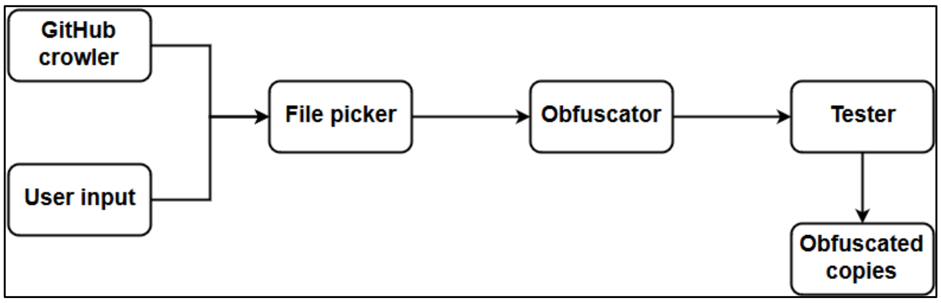


Figure 3. Design of PerfectPlaggie.

Testing of these obfuscated copies is very important – in theory if everything is done right, obfuscating will not change program behaviour, but we need to be sure. Thus, tester part will make sure, that all unit and integration tests pass.

4.2 Supported obfuscation types

Most important is to hide from computer, that source code is plagiarized. But also important thing is to think about that there is also possibility that some human expert will be also reviewing this obfuscated source code. For example, totally random variable names are very suspicious and therefore could trigger deep control of this code. Most important planned supported obfuscation types are described below.

Comments – there are multiple possibilities what to do with comments in source code. But we need to be very careful – because writing of comments is not so strict as writing source code, so any similarity in comments is considered as very suspicious. The safest option is too delete all comments.

Source code formatting – source code will be formatted according to programming language conventions – so even if two codes have exactly same formatting it is not suspicious – because it is convention.

Renaming of variables, methods, classes – random variable names or slight modification of original names is too suspicious for human experts. So there need to be some kind of synonyms dictionary for variables.

Conditions – the simplest method is to negate all conditions, but it is too easy to detect. Reordering of parts in composition conditions seems like a headache for plagiarism detection systems.

Line reordering – one of most effective methods for obfuscating, but also one of most difficult to implement – need to be implemented with help of PDG (Program dependence graph).

Splitting/merging of methods - another relatively simple and effective obfuscation. But we need to be careful with too much splitting or merging – there needs to be balance to be safe from human experts detection.

Wrapping – effective and simple. Plus, wrapping also adds a lot of extra lines of codes – so it is lowering similarity percentage by this way too.

Adding of redundant code – this can get very tricky. Because this added code must look and must do similar tasks like original, otherwise it could get really suspicious.

5 Conclusions

This paper proposes design of source code obfuscating system called PerfectPlaggie. This tool is designed to construct autonomously big datasets of obfuscated (plagiarized) copies from freely available source codes.

The hypothesis that such a tool can be constructed is supported by perfect plagiarism experiment. This experiment shows that it is not too complicated to create “perfect plagiarism” for human, also it is not much time consuming.

This tool is unique – because it works directly on transformation of source code (not bytecode), it is autonomous and user can choose what obfuscations he wants to be applied to original source code.

Because this is proposal of such a tool, there needs implementation to be done in future, to fully confirm or reject the hypothesis. Also I see potential in research for new obfuscation types, but we need to create them that way, that even human experts will not get suspicious – and this is really challenging task.

Acknowledgement: This contribution is the partial result of the project Research of methods for acquisition, analysis and personalized conveying of information and knowledge, ITMS 26240220039, co-funded by the ERDF.

References

[1] Arwin, C., Tahaghoghi, S.M.M.: Plagiarism Detection across Programming Languages. In *Twenty-Ninth Australasian Computer Science Conference* (ACSC2006). (2003). pp. 10.

[2] Ahtiainen, A., Surakka, S., Rahikainen, M.: Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises. In *Proceedings of the 6th Baltic Sea conference on Computing education research*: Koli Calling. (2006). , pp. 141–142.

[3] Bowyer, K.W., Hall, L.O.: Experience using MOSS to detect cheating on programming assignments. In *Frontiers in Education Conference 1999 FIE99 29th Annual*. (1999), vol. 3, Piscataway, NJ, United States, pp. 13–18.

[4] Chudá, D. Návrat, P., Kováčová, B., Humay, P.: The issue of (software) plagiarism: A student view. In *IEEE Transactions on Education*. (2012), vol. 55, no. 1, pp. 22–28.

[5] Fiducia, N.: *When Two Worlds Collide: The Oracle And Google Dispute*. Mondaq.com. (2013). Available at: http://www.mondaq.com/unitedstates/x/271942/ [Accessed: 14 Feb 2016].

[6] Joy M., Luck, M.: Plagiarism in Programming Assignments. In *IEEE Transactions of Education*. (1999), vol 42, no. 2, pp. 129-133.

[7] Juan, A.C.: Studying the Impact of Obfuscation on Source Code Plagiarism Detection. January, (2014), pp. 1–39.

[8] Lafortune, E.: 'ProGuard' Proguard.sourceforge.net. (2013), [online] Available at: http://proguard.sourceforge.net/ [Accessed: 18 Feb 2016].

[9] Prechelt, L. Malpohl, G., Philippsen, M.: Finding Plagiarisms among a Set of Programs with JPlag. In *Journal Of Universal Computer Science*. (2002). vol. 8, no. 11, pp. 1016–1038.

[10] Schleimer, S., Wilkerson, D.S., Aiken, A.: Winnowing: Local Algorithms for Document Fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data* - SIGMOD '03. (2003), pp. 76–85.

[11] Tahir Ali, A.M. EL, Abdulla H.M.D., Snášel, V.: Overview and comparison of plagiarism detection tools. In *CEUR Workshop Proceedings*. (2011). vol. 706, pp. 161–172.

[12] yWorks GmbH. yGuard - Java™ Bytecode Obfuscator and Shrinker. (2016). Available at: https://www.yworks.com/products/yguard [Accessed: 18 Feb 2016].

Similarities in Source Codes

Marek ROŠTÁR*

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
rostarmarek@gmail.com

Abstract. With an increasing popularity of different programming languages, a problem of finding similar parts of source codes across different programming languages is rising. Finding such parts of codes can be useful for improving source code quality or identifying potential plagiarism. In current day and age there are multiple ways of identifying similarities in the source code or text documents. Most known are text/token based methods, which can be strengthened with stronger preprocessing of given source codes. In this work we focus mainly on identifying similarities using abstract syntax tree. We also explore the possibilities of applying different levels of preprocessing of source code and its benefits from the performance point of view.

1 Introduction and related work

In current day and age with the development of software there is an increase of problems concerning plagiarism, but also it is quite common to see repeated use of source code parts. These two issues are the main reasons to explore the task of source code comparison, since it can detect most of plagiarism attempts and also highlight which parts of our source code are we using repeatedly. Such parts we may consider to turn into some sort of library or plugin, to improve our source code quality.

Plagiarism is these days one of the relevant problems of the academia, affecting not only text documents but also most of intellectual property including source codes. It is common that students inspire themselves with some work they found on the Web.

In this work we focus on detecting plagiarism in source code specifically (considering its special features in comparison to the standard text documents). Methods used to detect plagiarism in source code differ from methods used to detect plagiarism in text documents [1, 2], since the text in source code does not carry only meaning but some sort of function as well.

Plagiarists try to deceive anyone viewing their work with a multitude of different plagiarism attacks, which for plagiarism in source codes, the basic ones are as follows: altering comments in source code, altering whitespaces present in source code, altering names of variables, altering the order of parts of the source code, altering algebraic expressions in source code, translating source code into another programming language and extracting parts of source code.

[*] Bachelor degree study programme in field: Informatics
Supervisor: Dr. Michal Kompan, Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies STU in Bratislava

While altering comments in source code, the plagiarist leaves the whole structure of the source code and its functionality intact, but alters the visual part of the source code by either adding, removing or changing the content of the comments. Comments usually bear no functionality in the source code and usually are used to explain the source code.

Altering whitespaces is very similar to altering comments, but it is worth to note that this type of attack is not always applicable, since there are some programming languages in which whitespaces bear functionality. One of such programming languages is Python, in which whitespaces are used to denote blocks. Altering the names of variables will as well change the visual structure of the source code while not interfering with its functionality.

While altering the order of parts of the source code the plagiarist will change the order of separate blocks of source code. This does not interfere with the functionality of the source code and changes it visual structure. It is worth noting that in some languages order of functions matters. In these languages if some function will be using another one that is in original source code written sooner and in the altered source code the order is reversed, the function used in the other function has to be declared before it is used.

Altering algebraic expression takes advantage of commutative and distributive properties of certain algebraic operations. There is also the possibility of changing some parts of expressions that compare two numbers in a way that it will have the same logic but it will be visually different.

Translating source code to different programming language is not always a plagiarism attack, since sometimes the language to which is the source code being translated to does not have all the tools and structures necessary being used in the original programming language. In our work we focus on such sort of translation, which does not include any additional work from the person that translated the source code.

Extracting parts of the source code means that the plagiarist will remove some part of the source code and put it into a separate file. He will then include in the source code and call the functionality removed from original source code from the external file, and thus changing the visual structure of the source code while not interfering with its functionality.

The comparison of different plagiarism detection methods over various plagiarism attacks was discussed in [2], where the authors tested performance of different plagiarism detection methods fare against different types of plagiarism attacks on small scale source codes.

In [1], different types of plagiarism are discussed as well as methods used to detect plagiarism in general not only in source codes. This work goes over the differences in the literal plagiarism and intelligent plagiarism.

There are some works that explore combining different methods of detecting plagiarism [7, 8] to create so-called hybrid methods of plagiarism detection. In these specific works combination of tree based methods with either semantic based methods or token methods is applied.

Park et al. [5] discusses the application of source code abstraction for large scale source codes to improve the efficiency and accuracy of detecting similarities in those source codes. It proposes an automated abstraction method, which gets rid of large part of the source code, which is deemed as unimportant. Chillowicz et al. [3], improves method based on abstract syntax tree by creating fingerprints of compared documents and comparing them.

2 Plagiarism detection methods

There are several views used to detect plagiarism hierarchies. In this work we will differentiate four basic approaches [6]:

- Text based methods
- Token based methods
- Tree based methods
- Semantic based methods

Another example of method division into groups is where we divide methods into two groups depending on the fact if they take into consideration the meaning of the part they are comparing or not. In this view, we would put only text based methods in the group that does not take into consideration the meaning of source code.

2.1 Text based methods

Text based methods are generally (in the context of source code plagiarism) the quickest of all of the types of plagiarism detection methods. On the contrary, these are also most vulnerable to plagiarism attacks. Text based methods go step by step through the source code while comparing strings in them. Such methods can return various metric depending on which algorithm belonging to this type is used.

One of such algorithm is LCS algorithm [8], which returns the longest common subsequence between both source codes that are being compared through this algorithm.

The advantages of using text-based methods are that they are fast, they don’t require complicated data structures and are generally easier to understand. The disadvantages are that they are vulnerable to attacks, which means they often need to have the source codes heavily pre-processed to counter these plagiarism attacks, and they do not take into consideration the meaning and context of parts of the source code.

2.2 Token based methods

Token based methods are based on the serialization of parts of the source code into tokens which in the next phase replace the actual source code. This tokenization is done by the mean of lexical analysis of the source codes before comparing them, and selecting important parts which are then turned into tokens.

Lexical analysis can be done for example by creating a finite-state machine, to which we will define rules containing regular expressions. Using such lexical analysis, it is easy to see that through the rules we can select parts of the source code that we find important and we can discard the rest. After we have selected the important parts, we compare these altered source codes using text based methods.

Advantages of token based methods are that they are more resistant plagiarism attacks than text based methods, while keeping close in their computation time to them. Disadvantage compared to text-based methods is slightly slower computation time and the addition of some data structures in lexical analysis.

2.3 Tree based methods

Tree based methods are based on converting source code into a data structure of tree and after the conversion comparing the trees instead of source codes themselves.

These methods usually create trees by doing lexical analysis, similar to token based methods. But afterwards they use the source code after the lexical analysis in a syntactical analysis, that takes blocks of source code and transforms them into nodes adding information about them, like position in source code. After the transformation is done the hash function is used to compute hash values of given nodes. Then the comparison of the trees on a node to node basis is performed. Nowadays there are parser we can use to do the lexical and syntactical analysis, instead of having them as a separate part.

Advantages of this method are that it is resistant to plagiarism attacks and it can find behavioral changes, which can be used for finding malicious source code [4]. Its disadvantages are time consumption and addition of new data structures.

2.4 Semantic based methods

Semantic based methods do take into account not only the meaning of the source code, but also the context of parts of the source code. Thanks to this it can handle polysemy and synonymy. One of such semantic based methods uses program dependency graphs, in a similar way as the tree based methods use trees. Source codes are converted into graphs and then to compute similarity metric, we need to convert the graphs into adjacency matrices. When the matrices differ in size the smaller one extended with rows and columns of zeros to have the same size as the bigger one. Then we need to convert the matrices into vector and by comparing those, we get the similarity metric.

3 Source code abstraction

When we are working with source code we can remove for us unimportant parts and thus create an abstract source code. We use this sort of pre-processing often when we compare source code to find similarities in given source codes. There are multiple levels of abstraction, and we can determine how strong abstraction we need based on how big is the source code [5]. For small and medium scale source codes we often use only low levels of abstraction, but for large scale source codes we often use strong levels of abstraction. Low levels of abstraction commonly remove comments, unifies whitespaces, unless it is written in a programming language in which whitespaces denote blocks. Stronger levels of abstraction can alter different things, such as removing declaration of variables, values of strings or algebraic expressions from the source code.

Thanks to abstraction, we can vastly reduce the volume of the source code and with that speed up the process of comparing the source code. It can also remove some false positive cases, but it can also create other false positive, by removing important parts of the source code that we did not think were important.

4 Proposed solution

In this work we propose method of detecting plagiarism using abstract syntax tree so let’s go now go into detail about this method. Our method can be divided into 5 steps: construction of the abstract syntax tree, computing hash values of nodes of the abstract syntax tree, adding information about node, comparing the trees and computation of similarity metric.

To create abstract syntax tree of the source code we use the parser. Thanks to this step we get rid of some unimportant parts of the source code. After the tree is completed we compute hash values of the nodes, where for leaves the value will be the hash of their content and for other nodes the hash value is the sum of the hash values of its children and its hash value. This eliminates plagiarism attacks that alter the order of source code, because even if we swap two children of a node, value in the node will remain unchanged. When we have computed hash values of the nodes of the tree we can add additional information about notes such as number of children nodes or type of node, which speed up the comparison process. This step is optional. Then we can compare the trees on a node to node basis and after we finish we compute similarity metric (e.g., cosine similarity, Jaccard index).

Proposed approach is designed to include different levels of abstraction on source code before we convert the source code into the tree. There are in total three levels of abstraction, ranging from removing only comments and unifying white spaces to the highest level, in which we replace strings values, remove variable declarations, unify the names of variables and greatly reduce the volume of the source code.

4.1 Evaluation

In the experimentation phase we will compare the computation time for comparison between the different levels of abstraction and performance of proposed approach (in the context of detected similarities and codes). In order to compare our results to the state-of-the-art solutions we also compare our method with results of MOSS, which uses token based method. For the comparison we will use dataset obtained from bachelor course Artificial intelligence on our faculty and the SSID dataset[1].

5 Conclusions

The problem of plagiarism is on the rise in the academia. It affects not only text documents but it spread its influence over different intellectual property such as source codes. Plagiarism of source codes differ from plagiarism of text documents, for example in attacks which plagiarist takes to obscure the fact that they have stolen given intellectual property, which we have discussed.

Currently there are multiple methods of detecting plagiarism in source codes. We have divided these methods into four groups of text based, token based, tree based and semantic based methods of detecting plagiarism. These methods take different time to compare source codes and are differently vulnerable to plagiarism attacks.

We can reduce amount of the time needed to compare source code by using abstraction on them to remove unimportant parts of the source code at the risk that we may lose some information when removing parts of the source code.

In our experiment we compare if using stronger abstraction on source code before sending it into abstract syntax tree and comparing the trees is viable. We use three different levels of abstraction and we compare the performance of our algorithm to a free source code comparison tool, which used token based method to detect plagiarism.

References

[1] Alzahrani, S. M., Salim, N., & Abraham, A.: Understanding plagiarism linguistic patterns, textual features, and detection methods. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, (2012), vol. 42, no. 2, pp. 133–149.

[2] Beth, B.: A Comparison of Similarity Techniques for Detecting Source Code Plagiarism. (2014).

[3] Chilowicz, M., Duris, E., & Roussel, G. Syntax tree fingerprinting for source code similarity detection. *IEEE Int. Conference on Program Comprehension*, (2009), pp. 243–247.

[4] Neamtiu, I., Foster, J. S., & Hicks, M.: Understanding source code evolution using abstract syntax tree matching. *ACM SIGSOFT Software Engineering Notes*, (2005), vol. 30, no.4, 1.

[5] Park, S., Ko, S., Choi, J. J., Han, H., & Cho, S.-J.: Detecting Source Code Similarity Using Code Abstraction Categories and Subject Descriptors. *Proc. of the 7th Int. Conf, on Ubiquitous Information Management and Communication - ICUIMC '13*, (2013), pp 1–9.

[6] Tao, G., Guowei, D., Hu, Q., & Baojiang, C.: Improved Plagiarism Detection Algorithm Based on Abstract Syntax Tree. *Emerging Intelligent Data and Web Technologies (EIDWT), 2013 Fourth Int. Conf. on*, (2013), pp. 714–719.

[7] Wu, S., Hao, Y., Gao, X., Cui, B., & Bian, C.: Homology detection based on abstract syntax tree combined simple semantics analysis. *Proceedings - 2010 IEEE/WIC/ACM Int. Conf. on Web Intelligence and Intelligent Agent Technology - Workshops, WI-IAT 2010*, (2010), pp. 410–414.

[8] Zhang, Y., Gao, X., Bian, C., Ma, D., & Cui, B.: Homologous detection based on text, Token and abstract syntax tree comparison. *Proc. 2010 IEEE Int. Conf. on Information Theory and Information Security, ICITIS 2010*, (2010), pp. 70–75.

[1] http://wing.comp.nus.edu.sg/downloads/SSID/

Synthesis of Hardware Systems Power Management

Miroslav SIRO*

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
`siro.miroslav@gmail.com`

Abstract. Power consumption is a problem in highly integrated hardware systems, mainly because of the rising power density and associated overheating of the chip. Therefore, the power must be managed in such hardware designs. However, the standard way of power management is quite complex and a way to simplify it is to use more automation in the design process. Our work is focused on automatic generation of the power-management specification of hardware systems at the register-transfer abstraction level (RTL). In this paper, we describe the design and implementation of a tool, called PMS2UPF, which automatically transforms abstract power-management specification from the SystemC standard, enhanced by the PMS library, to the more-detailed UPF (Unified Power Format) standard language. The automation of this transformation accelerates the design process, specifically the transition from the system level of abstraction to the RTL.

1 Introduction

Nowadays, the power consumption is one of the key considerations in hardware system development. It is very important in various kinds of hardware systems, like mobile systems where battery life is essential, servers where power costs are not negligible or general highly integrated hardware systems, where overheating is a problem. Therefore, need for power management arose. There are multiple approaches to hardware system design that have power management in mind, like built-in features of hardware description languages (HDL), external libraries for these languages or completely new languages dedicated to power management specification. The most well-known language for this purpose is UPF (Unified Power Format), [1]. These approaches provide various methods of power management, like clock gating, voltage and frequency scaling, power shut-off, operand isolation and others.

In this paper, we focus on automation of power management in hardware system design using SystemC standard and UPF language. There are multiple SystemC [2] libraries that handle power monitoring, estimation, modelling or management. Some of them are TLM Power 3 [3], PK

[*] Master degree study programme in field: Software Engineering
Supervisor: Dr. Dominik Macko, Institute of Computer Systems and Networks, Faculty of Informatics and Information Technologies STU in Bratislava

Tool [4], Power SC [5], Power Modelling Framework [6] PwARCH [7], or PMS [8]. We focus particularly on the PMS library.

PMS is a SystemC library that provides methods of abstract power management specification. Its central idea is the same as of the UPF language and that is a division of the hardware system into power domains.

The power domain [1] is one of the most basic concepts of power management. It is a group of modules of a hardware system that are powered in the same manner. These modules are usually located physically close to each other and powered by the same supply nets. If a component is a part of the power domain, then all its subcomponents are part of it too. Power domains are hierarchically structured and one power domain can be a part of the other. Every domain has one power state that is active and a list of power states that the domain can possibly achieve. Also, it is necessary to create a power state table, which specifies, which combinations of power states of various power domains are allowed [9].

In this paper, we propose a tool named PMS2UPF which automatically converts power management specification specified by SystemC design and functions of PMS library to commands of the more-detailed UPF language, which is an internationally recognized standard of power management specification. This automation significantly accelerates development process and reduces risks of introducing a human error to the design.

In the next section we introduce a work related to the problem area of our research. In Section 3 we describe a proposed transformation process, including the analysis of the input files with SystemC design and PMS specification, internal data model and generation of the UPF specification. In the last section, we make the conclusions from this work and plans for further work.

2 Related work

As mentioned before, there are numerous SystemC libraries that enable power monitoring, estimation, modelling or management. We introduce each of the mentioned libraries in this section.

TLM Power 3 [3] provides classes that allow a hardware system designer to estimate power consumption of the designs. The library requires the designer to modify classes of the design in a way to inherit from the basic class of TLM Power 3: *pw_module*. The TLM Power 3 uses two approaches towards power estimation: estimation by regime of individual modules or estimation by power per transaction.

Another mentioned library is PK Tool [4], which allows a designer to estimate power consumption of the design. Basic class of the library is *power_module*. An instance of this class is created for every module of SystemC design. PK Tool allows the designer to use models of power consumption to make predictions. It also allows user to specify custom power consumption models. Another feature of this library are *Augmented Signals* which can extract dynamic data from the signals.

PowerSC [5] is a library that allows a designer to monitor power consumption. It monitors a switching activity in the system design and provides macros to extract necessary data from the design.

These three libraries allow designer to monitor and estimate power consumption but they do not provide functions to manage it. These estimations might however become less accurate when power management is specified.

Next three libraries, Power Modelling Framework, PwARCH and PMS however allow the designer to specify power management.

Power Modelling Framework [6] allows user to model the power consumption of the designed system. It allows user to specify and change the power state of the system by using function *new_power_phase(phase).* This library also allows the designer to use Dynamic Voltage and Frequency Scaling (DVFS) by using function *new_power_mode(mode*).

PwARCH [7] allows the designer to model and verify power management specification. It adopts multiple concepts from UPF and abstracts them to the transaction layer. It supports power domains, power switches, and global power state table. After designing power management specification, a PMU (Power Management Unit) needs to be modelled as well.

Finally, there is the PMS (Power Management Specification) [8], which we have mentioned before. This library is inspired by UPF, but allows the designer to use system level of abstraction. Its main concept of power management specification is a power domain. It is however much easier to use than UPF. Benefit of this library is that it allows verification of power specification in the early stages of development which makes it easy and cheap to detect and correct errors.

We decided to focus our work on PMS library because it does not only allow to specify power management on the system level of abstraction, but also significantly simplifies it.

3 The proposed synthesis process

As stated above, PMS2UPF transforms a SystemC and PMS specification to the UPF specification. This transformation consists of three basic steps (See Figure 1).

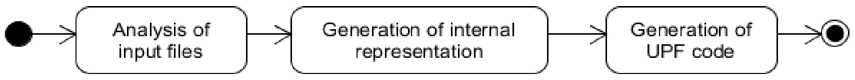
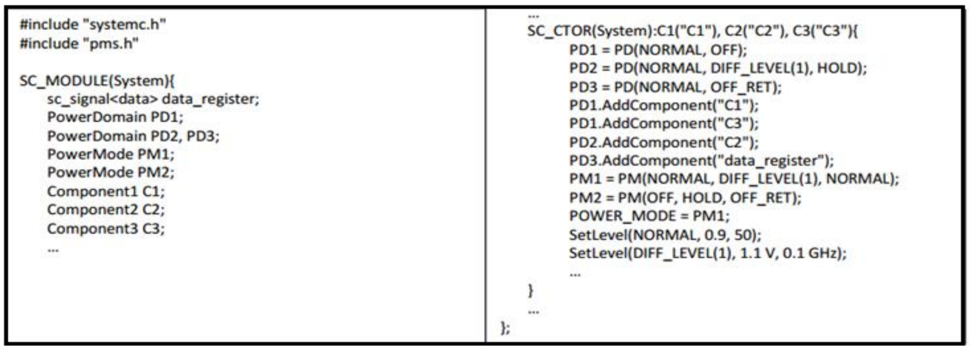


Figure 1. Three steps of transformation from PMS specification to UPF specification.

The first step is the analysis of input files, which loads all the specification files and identifies the important constructs in the design. The next step creates objects of important design structures and identifies relationships among them. The last step generates the power-intent specification in the UPF language based on the created internal object representation. These steps are executed sequentially. Each of these three steps mentioned above will be explained in more detail in the following sections.

3.1 Analysis of input files and creation of internal representation of the design

Analysis of input files requires parsing of SystemC source files mixed with PMS specification. For this task we have implemented custom parser of the code. Example of an input file with PMS specification can be seen in Figure 2.



```
#include "systemc.h"
#include "pms.h"

SC_MODULE(System){
    sc_signal<data> data_register;
    PowerDomain PD1;
    PowerDomain PD2, PD3;
    PowerMode PM1;
    PowerMode PM2;
    Component1 C1;
    Component2 C2;
    Component3 C3;
    ...
```

```
    ...
    SC_CTOR(System):C1("C1"), C2("C2"), C3("C3"){
        PD1 = PD(NORMAL, OFF);
        PD2 = PD(NORMAL, DIFF_LEVEL(1), HOLD);
        PD3 = PD(NORMAL, OFF_RET);
        PD1.AddComponent("C1");
        PD1.AddComponent("C3");
        PD2.AddComponent("C2");
        PD3.AddComponent("data_register");
        PM1 = PM(NORMAL, DIFF_LEVEL(1), NORMAL);
        PM2 = PM(OFF, HOLD, OFF_RET);
        POWER_MODE = PM1;
        SetLevel(NORMAL, 0.9, 50);
        SetLevel(DIFF_LEVEL(1), 1.1 V, 0.1 GHz);
        ...
    }
    ...
};
```

Figure 2. SystemC source file with PMS specification [8].

This parser opens all source files and reads module definitions from them. Each module is stored in separate string. This way we create mapping of one string to one module, which simplifies further analysis. Each of these strings is separated by characters *semicolon*, *opening* and *closing composed bracket* (;{}) to a list of smaller tokens. These tokens are then read multiple times and each iteration extracts some data from these tokens and creates objects representing these data. These data include sub-modules, ports, inner channels, connections between modules and PMS specification itself (example illustrated in Figure 2). Based on the extracted data we are able to create objects representing more-detailed UPF specification. Each object is going to represent one UPF command. Information about some of those objects is not explicitly specified by PMS specification therefore our tool needs to determine which objects need to be created. Examples of such objects are power switches, supply nets, isolation cells, level shifters, retention cells or power state table. Aim of this analysis is to create list of objects that contain all necessary information to create proper UPF specification. This process is illustrated in Figure 3.

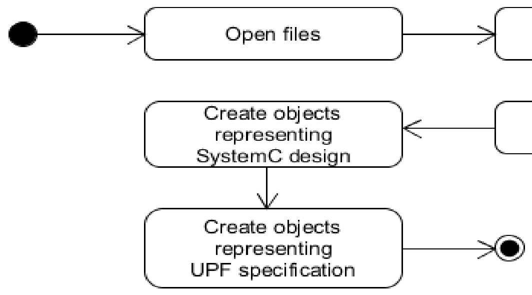

Open files
Load modules
Tokenisation
Create objects representing SystemC design
Create objects representing UPF specification

Figure 3. Process of analysis of SystemC source files.

During implementation of the analysis we have encountered several problems:

- A part of the information necessary to generate UPF commands is impossible to be extracted from the PMS specification alone. Therefore we have to analyse the SystemC design too.
- Names of the modules and ports are not known beforehand. Therefore we must search for declarations and definitions of the modules. We need to iterate over the tokens multiple times, which prolongs the analysis process. Implementation of this part is subject to further optimisation.
- The form of the input files (i.e. the designer’s coding style) is not known beforehand. Locations of whitespaces, number of lines and number of modules per file is not known. Therefore we had to create more robust solution that tokenises the files, excludes tokens that contain no relevant information and purify tokens of characters (whitespaces and others) that do not hold any useful information. This processing simplifies loading of all necessary data to create objects representing SystemC/PMS design and UPF specification.
- Not all tokens contain relevant information. Some of them might be comments or they describe parts of SystemC design that are not needed in our internal representation. Therefore we must be able to determine which tokens are necessary to analyse and which are not.

Internal data model that contains representation of SystemC design and UPF specification consists of three parts:

- Abstract model: Abstract model contains prototypes for each object that represent some part of SystemC design. For example each module type is represented by one object and serves as a basis to create every instance of the module.

- Specific model: Objects of specific model represent every instance of the module or other element of the SystemC design or PMS specification. These objects hold all information necessary for generating of the UPF specification.
- Auxiliary objects and classes: These objects have supplementary roles at creation of abstract or specific model. For example, there is a class that represents connection of port to the channel. These classes do not represent anything from SystemC design or UPF specification but they simplify the implementation.

3.2 UPF Generation

In this section, we describe a method of generating the UPF specification. The UPF generator sequentially iterates through the lists of objects representing UPF specification and generates a UPF command based on each object. Finally, all generated UPF commands are concatenated to one long string and written into a text file. However, we must adhere to the following constraints of the UPF language:

- We need to keep a correct order of the UPF commands. For instance, we need to create power domains at first and only then we can generate supply nets.
- Structure of some UPF commands is dependent on other UPF commands. For example the number of power switch control ports depends on the number of power states that corresponding power domain can achieve.

The following source code illustrates some parts of the UPF output of the PMS2UPF tool, namely power domains, some supply nets and one power switch. A complete UPF specification even when based on a simple PMS specification can, take hundreds of lines. Therefore we are not going to present full example of the output.

```
create_power_domain PD1 -elements { DUT.System.C1 DUT.System.C3 }
create_power_domain PD2 -elements DUT.System.C2
create_power_domain PD3

create_supply_port VSS
create_supply_net VSS -domain PD1
create_supply_net VSS -domain PD2
create_supply_net VSS -domain PD3
connect_supply_net VSS -ports VSS

create_supply_port sn_1
create_supply_net sn_1 -domain PD1
create_supply_net sn_1 -domain PD2
create_supply_net sn_1 -domain PD3
connect_supply_net sn_1 -ports sn_1

…

create_power_switch ps_PD1
 -domain PD1
 -input_supply_port { in0 VSS }
 -input_supply_port { in1 sn_1 }
 -output_supply_port { out PD1_V }
 -control_port { cp_6_0 cp_6_0_s }
 -off_state { OFF in0 { ! cp_6_0 }}
 -on_state { NORMAL in1 { cp_6_0 }}
```

In the code above, we can see that from information about power domains and power states, our tool generates supply nets and power switches. It also generates isolation cells, level shifter cells, retention cells and power state table. Those are however not shown in the example above.

4 Conclusions

In this work, we have proposed and implemented a tool that simplifies usage of the PMS library. The developed tool allows the designers using the PMS library to easily convert their power management specification to the UPF standard. This increases speed of development process and reduces the possibility of introduction of human errors into the design. The reason is that with the PMS2UPF tool a hardware system designer and PMS user does not need to transform the power management specification from SystemC to UPF manually. This significantly simplifies the validation and debugging process. Our tool provides a simple command line user interface which makes it easier to use in conjunction with other programs and thus to automate even larger portion of the design process.

In the future work we plan validate the resulting UPF specification by using the Modelsim [10] tool, so we can guarantee validity of the generated UPF code. We also plan to optimize our tool so the transformation process can run faster, create a graphical version of the user interface and provide more options to the user. For example, allow the system designer to specify which particular version of UPF (1.0 or 2.0) our tool should provide as an output.

Acknowledgement: This work was partially supported by the Slovak Scientific Grant Agency (VEGA 1/0616/14 “Methods for the design and verification of digital systems with low power consumption using formal specification languages”).

References

[1] IEEE: Standard for Design and Verification of Low-Power Integrated Circuits 1801. IEEE, (2013).

[2] IEEE: Standard for Standard SystemC Language Reference Manual 1666. New York, IEEE, (2011).

[3] Greaves, D. and Yasin, M.: TLM POWER 3: Models, Methods and Tools for Complex Chip Design, LNEE, (2014).

[4] Vece, G., Conti, M.: PKtool: Power estimation in SystemC. Available at http://www.deit.univpm.it/PKtool/

[5] Klein, F. et al.: SystemC-based power evaluation with PowerSC. Electronic system level design: An open-source approach, Springer, (2011).

[6] Lebreton, H. and Vivet, P.: Power Modeling in SystemC at Transaction Level, Application to a DVFS Architecture, Grenoble: IEEE Computer society, (2008).

[7] Mbarek, O., Pegatoquet, A. and Auguin, M.: Using unified power format concepts for power-aware design and verification of systems-on-chip at transaction level. IET Circuit Devices Syst Vol. 6. Nice, IET, (2012), vol. 6, pp. 287-296.

[8] Macko, D., Jelemenská, K. and Čičák, P.: Power-Management Specification in SystemC, IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits and Systems. IEEE, Bratislava, (2015), pp. 259-262.

[9] Power Format Initiative. A practical Guide to Low Power Design.: Power Forward, (2012).

[10] Mentor Graphics. mentor.com. Available at https://www.mentor.com/products/fv/modelsim/

Configurable Spare Database Reduction for RAMs

Marek SPURNÝ[*]

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
`spurny.marek@gmail.com`

Abstract. Built-in self-repair for memories is widely used in industry to increase their yield. Recently some approaches were proposed which use flexible configurable spares. However, the reconfiguration control is more complex as with traditional spares because there are many spare configurations that can be used to repair faults and the repair algorithm has to keep the database of these configurations. This database is generally very large which is not feasible in order to keep the area overhead acceptable. In this paper a method for configurable spare database reduction is proposed along with its hardware implementation.

1 Introduction

Current semiconductor process technologies cause the fault density in RAMs to increase. In present day system on a chips (SoCs) memories are dominating the chip area therefore the overall SoC yield is impacted mostly by memory yield. To increase memory yield, built-in self repair (BISR) is widely used in industry. Spare memory cells are addressed instead of faulty ones. The basic BISR scheme is shown in Figure 1. The built-in self test (BIST) block tests main memory for faults. If a fault is found, its location is sent to the built-in repair analysis (BIRA) block which determines which spare will be used to replace the fault. The final repair solution for all faults is stored in the address reconfiguration (AR) block. When a faulty cell is being addressed, the address is translated into the spare memory by AR and the spare is addressed instead of the faulty cell.

Most of existing approaches [1, 3] use the traditional spare model with rows and columns. The entire row or column which contains faulty cell(s) is replaced by a spare row or column. An example memory with 2 spare rows and columns is shown in Figure 2 (a). Spare S1 was used to replace a faulty row and spare S3 was used to replace a faulty column in the memory.

Recently some approaches were proposed which use spare model with more flexible configurable spares [4]. A spare can be configured as a traditional row or column but also as other shapes such as a square or a rectangle. These various configurations are referred to as *configuration types*. An example memory with 4 spares is shown in Figure 2 (b). Both used spares S1 and S2 were configured as rectangles and replaced the faulty areas of the corresponding shapes in the memory.

[*] Master study programme in field: Software Engineering
Supervisor: Dr. Štefan Krištofík, Institute of Computer Systems and Networks, Faculty of Informatics and Information Technologies STU in Bratislava

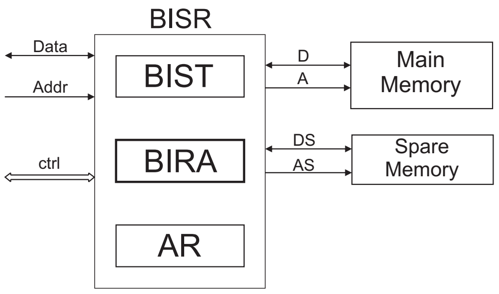


Figure 1. BISR for memories.

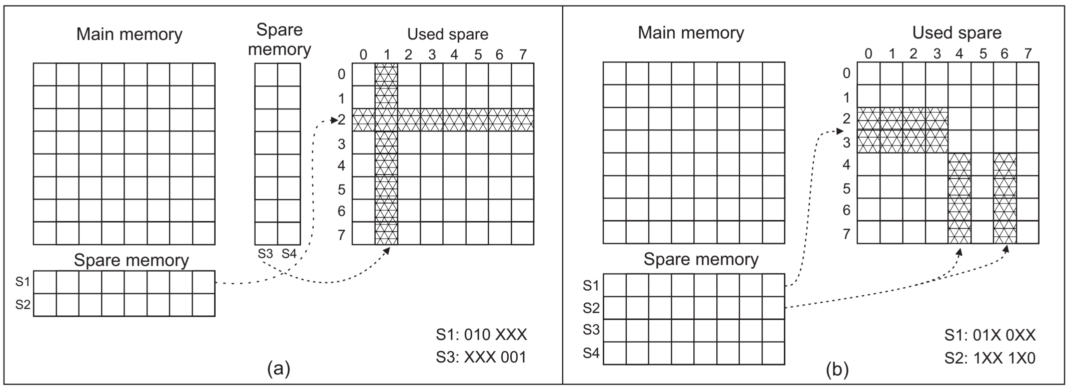


Figure 2. Spare architectures: (a) traditional, (b) configurable.

In Figure 2 the *address space* of each used spare is shown. This space contains the addresses of all faults it can possibly replace. For example, the spare row S1 in Figure 2 (a) can replace all faults in row 2 in the memory, therefore its address space is 010 XXX. Similarly, the address space of spare S2, which can replace all faults in column 1 is XXX 001. The address space of configurable spares can be obtained similarly. For example, the spare rectangle S1 in Figure 2 (b) can replace faults in rows 2 and 3 in the memory, but only in columns 0 through 3. Its address space is 01X 0XX.

Before the first fault is detected by BIST in the traditional spare model, it is clear that the fault could only be replaced by either row or column when it appears. However, in configurable spare model, the set of possible spare configurations that could replace the first fault can be large. The exact number of possible configurations is determined by the memory size.

In the configurable spare model, the repair algorithm has to keep a database of all configurations that can possibly repair the faults. The database is progressively reduced as more faults are detected [4]. This database can be initially very large (depending on the memory size) which can increase the BISR area overhead to unacceptable levels. To keep the area overhead feasible, a method for configurable spare database reduction is proposed along with its hardware implementation.

2 Configurable spare database reduction principles

In the configurable spare model as was shown in Figure 2 (b), before the first fault is detected by BIST any spare configuration could be used to replace it, because it is not known where the fault will occur. Therefore, if the database of spare configurations that could possibly replace the first fault (in the rest of the paper this will be referred to simply as 'database') is constructed before the first fault is detected, it has to contain all configurations. However, if the database is constructed after the first

Table 1. Database size.

		#areas	
		any	max. 2
#faults	0	1120	272
	1	70	17

fault is detected, its size can be narrowed down significantly.

In general, the database size can be reduced M times (M being the bit size of each spare) when it is constructed after the first fault is detected as opposed to when it is constructed before the first fault is detected. The reason behind this is that the incoming address of the first fault rules out the other $\frac{M-1}{M}$ configurations of each type which cannot be used to replace the fault. For example, suppose that the first fault detected in the memory in Figure 2 is (2,3). The row address of the fault is 2, therefore it can only be replaced by replacing row 2 with a spare row. Other $\frac{7}{8}$ row configurations are ruled out. Same principle is true for any spare configuration type (column, square, rectangle).

The database can be reduced further if only a subset of all configuration types is considered, i.e. only spare configurations with a set maximum number of areas. An *area* of a spare is a continuous block of spare cells. For example, spare S1 in Figure 2 (b) has 1 area and spare S2 has 2 areas.

The database reduction method proposed in this paper is based on the following principles:

1. The database is constructed after the first fault is detected.
2. Only spare configurations with a maximum of 2 areas are considered.

3 Implementation

The proposed configurable spare database reduction method was implemented for a small 1 kB memory (64x16 bits) with 8 16-bit spares. 5 spare configuration types are considered: row (1x16 bits), column (16x1 bits), square (4x4 bits), horizontal rectangle (2x8 bits) and vertical rectangle (8x2 bits). Table 1 shows the database sizes for this memory in various cases if using or not using principles 1 and 2 mentioned in Section 2. The initial size of the database when not using the principles is 1120. This means the first fault could be replaced by 1120 various spare configurations. Some reduction can be achieved by using only one of the principles, but if both principles are used, the database size can be reduced to 17 entries. This means there are only 17 spare configurations that could possibly repair the first detected fault.

Per principle 2, there are only spare configurations with a maximum of 2 areas in the reduced database. The configurations with 1 area include 1 row, 1 column, 1 square, 1 horizontal rectangle and 1 vertical rectangle configuration. The configurations with 1 area are referred to as *default*. To obtain the remaining configurations with 2 areas, the operations *RshiftN* and *CshiftN* are defined for the default square and both rectangle configuration types, as follows:

- RshiftN - the leftmost 'X' bit in the row address of the address space of the spare is shifted N positions to the left.
- CshiftN - the leftmost 'X' bit in the column address of the address space of the spare is shifted N positions to the left.

In both operations, the incoming fault address bit is inserted in the original bit position of the shifted 'X' bit. These operations are not defined for row and column spare configurations. An example of Rshift and Cshift operations for a square spare configuration (4x4 bits) is shown in Figure 3 and the following is true:

- `10XX 10XX` - address space of the default square configuration.
- `1011 1010` - address of the first detected fault (11,10).
- `1X1X 10XX` - address space produced by the Rshift1 operation.

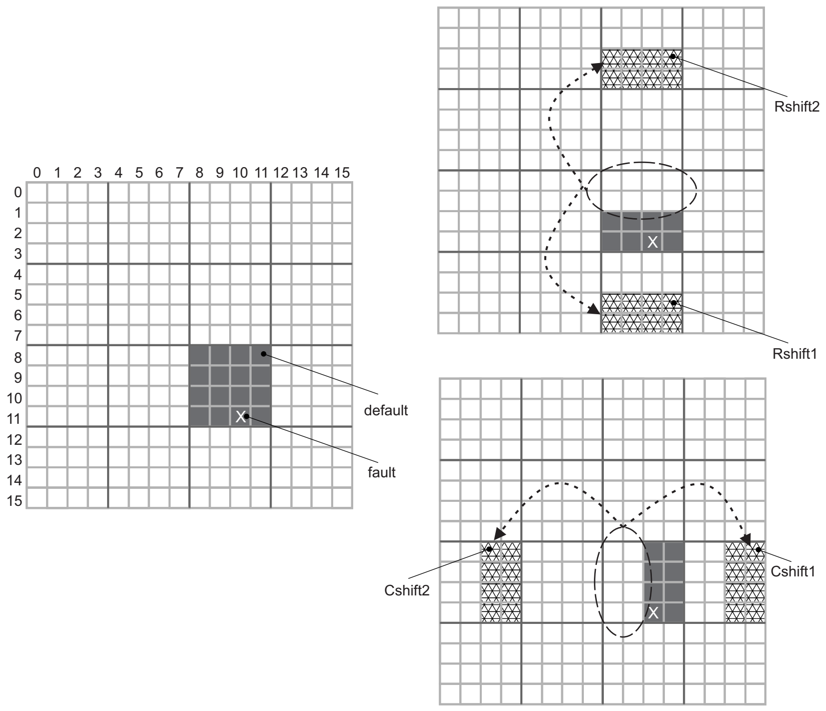


Figure 3. Rshift and Cshift operation examples for a 4x4 spare configuration.

- `X01X 10XX` - address space produced by the Rshift2 operation.
- `10XX 1X1X` - address space produced by the Cshift1 operation.
- `10XX X01X` - address space produced by the Cshift2 operation.

Similarly, the Rshifted and Cshifted address spaces are produced for the horizontal and vertical rectangle spare configurations and the reduced database is constructed. All entries of the reduced database are shown in Table 2. The database has 17 entries, comprising 1 row, 1 column, 5 square, 5 horizontal rectangle and 5 vertical rectangle configurations including all default, Rshifted and Cshifted entries. Note that the database is generalized for any fault, therefore the incoming fault row address bits are noted as $R_1, \ldots, R_4$ and the column address bits as $C_1, \ldots, C_4$.

4 BIRA hardware implementation

The first fault in the 1 kB memory described in Section 3 can be replaced by 17 spare configurations. The subsequent detected fault can further reduce the set of configurations that could possibly repair both faults. In the worst case, none of the 17 configurations could repair both faults. In this case the next spare has to be activated and used to repair the second fault. On the other hand, several faults may be detected that could still be repaired using the first spare.

To progressively rule out the various spare configurations that could no longer repair the incoming faults, for each spare one FGEN circuit shown in Figure 4 is used. The address of the first detected fault (A1) is compared with all subsequent addresses of detected faults (AN) by the CMP block. 17 comparisons are performed by the CMP block to check which of the 17 spare configurations in the database (according to Table 2) could still be used to repair the incoming faults. The flag bits (the first column in Table 2) are set to 0 by the MEM block for configurations that can no longer repair faults and to 1 for configurations that can still repair faults. The flag bits are updated on the rising edge of the *update* signal but only if the *det* signal is active. One AND gate is used for this purpose.

Table 2. Reduced database.

bit flag #	type	dim.	row address				column address			
1	default	1x16	R_1	R_2	R_3	R_4	X	X	X	X
2	default	16x1	X	X	X	X	C_1	C_2	C_3	C_4
3	default	4x4	R_1	R_2	X	X	C_1	C_2	X	X
4	Rshift1		R_1	X	R_3	X	C_1	C_2	X	X
5	Rshift2		X	R_2	R_3	X	C_1	C_2	X	X
6	Cshift1		R_1	R_2	X	X	C_1	X	C_3	X
7	Cshift2		R_1	R_2	X	X	X	C_2	C_3	X
8	default	2x8	R_1	R_2	R_3	X	C_1	X	X	X
9	Rshift1		R_1	R_2	X	R_4	C_1	X	X	X
10	Rshift2		R_1	X	R_3	R_4	C_1	X	X	X
11	Rshift3		X	R_2	R_3	R_4	C_1	X	X	X
12	Cshift1		R_1	R_2	R_3	X	X	C_2	X	X
13	default	8x2	R_1	X	X	X	C_1	C_2	C_3	X
14	Rshift1		X	R_2	X	X	C_1	C_2	C_3	X
15	Cshift1		R_1	X	X	X	C_1	C_2	X	C_4
16	Cshift2		R_1	X	X	X	C_1	X	C_3	C_4
17	Cshift3		R_1	X	X	X	X	C_2	C_3	C_4

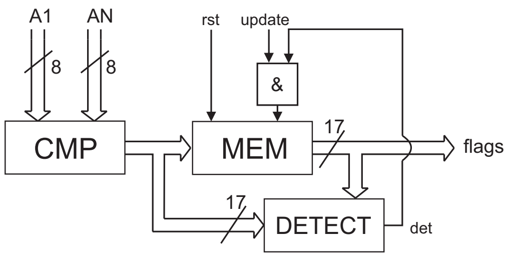


Figure 4. Flag bit generator - FGEN.

The DETECT block detects a possible situation when all flag bits would be set to 0 (indicating a new spare is needed) and prevents this from happening by setting the *det* signal to 0. This is important to keep the information on the flag bits active even when a new spare is needed.

The overall scheme of the proposed BIRA architecture for all 8 spares is shown in Figure 5. It contains one FGEN circuit and one A1GEN circuit for each spare and the BIRA CONTROL block. FGEN circuit was already described. A1GEN circuit is used to distinguish between the first fault and the other faults and generates the first fault address A1 for the corresponding FGEN circuit. A1 is generated on the rising edge of the *update* signal but only if the corresponding *ff* signal is active. The *ff* signals indicate that the incoming fault is the first for the corresponding A1GEN circuit. The BIRA CONTROL block controls the whole process by generating the appropriate values of *rst, update (upd.)* and *ff* signals. Incoming fault addresses from BIST are sent to BIRA through the 8-bit *A* signal.

After the repair process is finished, the final repair solution is indicated on the flag bits of FGEN circuits.

5 Conclusions and future work

The proposed BIRA architecture is based on the CSA algorithm [4] and utilizes a new method for database reduction. It is a part of the BISR architecture which contains a BIST block capable to

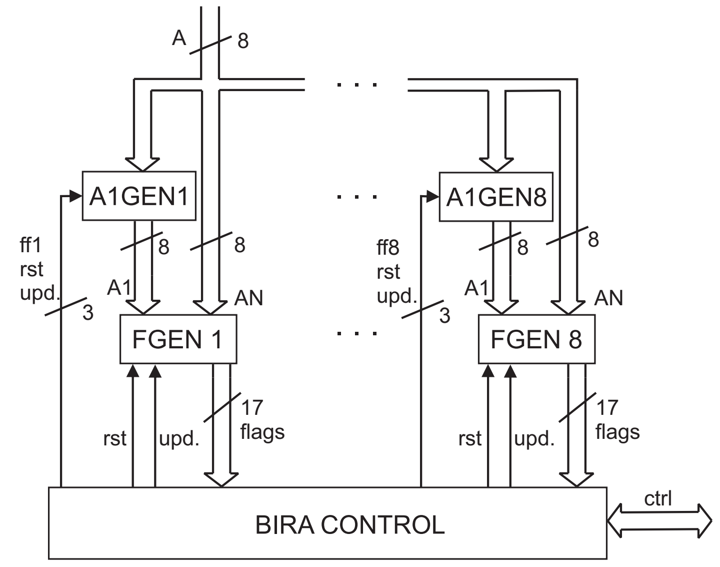


Figure 5. Proposed BIRA architecture.

test the main memory with MARCH C- and MARCH B test algorithms [2], 1 kB (64x16 bits) main memory and 8 16-bit spares.

Future work includes the design of the BIRA control block as well as the AR block. The control block will generate the necessary control signals as was described in Section 4. The AR block will process the final repair solution stored on the flag bits of FGEN circuits as was described in Section 4.

The entire BISR architecture will be described in the VHDL language and will be fully synthesizable. The area overhead will be evaluated for various memory sizes and number of spares to verify its feasibility.

Acknowledgement: This work has been supported by Slovak national project VEGA 1/0616/14.

References

[1] Chen, T.J., Li, J.F., Tseng, T.W.: Cost-Efficient Built-In Redundancy Analysis With Optimal Repair Rate for RAMs. *Trans. on Computer-Aided Design of Integrated Circuits and Systems, IEEE*, 2012, vol. 31, no. 6, pp. 930–940, doi: 10.1109/TCAD.2011.2181510.

[2] Fischerova, M., Gramatova, E.: *Chapter 7: Memory Testing and Self-Repair, In: Design and Test Technology for Dependable Systems-on-Chip*. IGI Global, Hershey, Pennsylvania, 2010, doi: 10.4018/978-1-60960-212-3.

[3] Jeong, W., Kang, I., Jin, K., Kang, S.: A Fast Built-in Redundancy Analysis for Memories with Optimal Repair Rate Using a Line-Based Search Tree. *Trans. on VLSI Systems, IEEE*, 2009, vol. 17, no. 12, pp. 1665–1678, doi: 10.1109/TVLSI.2008.2005988.

[4] Lee, M., Denq, L.M., Wu, C.W.: A Memory Built-In Self-Repair Scheme Based on Configurable Spares. *Trans. on Computer-Aided Design of Integrated Circuits and Systems, IEEE*, 2011, vol. 30, no. 6, pp. 919–929, doi: 10.1109/TCAD.2011.2106812.

ISBN 978-80-227-4551-2

9 788022 745512