STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LX**, Number 1, 2015

USING CONCERN SPACES TO MEASURE REQUIREMENTS SIMILARITIES

CĂLIN EUGEN NICOLAE GAL-CHIŞ

Abstract. The software artefacts are crucial during the development cycle of a software product and tracing them is important to the development process. The model used, the requirements document, and the code, are artefacts that can be updated or reused in different projects. Different types of notations are used to add traceability to artefacts, providing versatility in searching, indexing, updating, or retrieving them.

MultiCoS is an approach based on separation of concerns (SoC) in multiple spaces. The concern spaces are defined by grouping concerns by common interest. The relationships between concerns and different types of entities empowers the concern to provide a degree of meaning to an entity. Defining and using concerns to properly describe software artefacts can add semantic to documents such as the specifications document, requirements document, project documents, and to other artefacts such as code files or modules. Given this, the concerns and their relationships can provide traceability to higher level entity spaces, such as the application model, the views, and the design documents of a software application.

The MultiCoS metamodel is presented here in a case study, reusing web applications artefacts in the software development cycle. In addition to other tracing methodologies, MultiCoS can add semantic value to artefacts and can strengthen the relationships to concerns or between artefacts by taking into account similarity coefficients.

In contrast to other methodologies, MultiCoS supports complex tracing systems by creating multiple relationships of different degrees between entities, based on the value of a concern, a subunitary value that measures the impact of a concern to an entity.

1. Introduction

Aspect Oriented Software Development (AOSD) [10] methodologies and Separation of Concerns (SoC) [15] approaches in software development are

Received by the editors: May 1, 2014.

2010 *Mathematics Subject Classification.* 68N30.

1998 *CR Categories and Descriptors.* D.2.1 [**Software Engineering**]: Requirements/Specifications – *Methodologies*; D.2.13 [**Software Engineering**]: Reusable Software – *Domain engineering*.

Key words and phrases. Concern Spaces, Requirements Engineering, similarities metric.

supporting and controlling the development process, in a way that provides a better understanding of the relations between concerned entities. Such a need is mentioned early in software development [3]. Leaded by the relations between concerned entities, the programmers can understand the impact of their work in the final product, and the stakeholders can trace easier how product specifications and requirements were implemented into the software product. MultiCoS separation of concerns approach offers a solution on coping with complexity and with cross-cutting concerns, also being able to provide support for code reusability and reverse engineering.

In terms of requirements specification, in the initial stages of the software development, every approaches is providing and using their own proprietary methods of describing the relations and dependencies between requirements. In addition to providing a method for managing requirements that is suitable to be used with any approach, MultiCoS can add a list of attributes to the requirements. These attributes are providing meaning to the requirements in the context of the software development process. Later in the development process, these atributes can also be used to describe artefacts such as the project documents, the code and other entities interacting with the software product. In fact, these attributes are the *concerns*, and these concerns are grouped into *concern spaces* by their matter of interest.

Some concern spaces will address generic attributes, while others will be specific. A set of default Concern Spaces can be used to properly describe the basics related to the software developement process. Other standard libraries of concern spaces are meant to describe different aspects of the software product and of the parties involved in the software development process. Custom, new concerns can be created and used as needed to fully describe the desired attributes, and concern spaces can be set to encapsulate these new concerns.

In this paper we will show that, while there are certain modelling approaches using concerns to describe specific entities and relations in the software development process, the MultiCoS approach uses a general-purpose concern modelling capability to support the software development process in various aspects and also provides a proper environment for code reusability or software reengineering.

In Section 2 will be presented the existing methodologies based on different approaches using separation of concerns. In Section 3 the MultiCoS approach will be presented, with the primitives used here regarding Concern Spaces, a mapping function used to relate different artefacts and entities to Concern Spaces. This paper's main contribution: a metric and a process that will be used over the mappings to determine artefacts similarities, will also be introduced in Section 3. In Section 4, a Study Case will be conducted. In the end, there will be presented the directions for further work and the conclusions.

2. Related Work

The main venues related to this research paper are the methodologies based on SoC and AOSD. They have introduced the core principles of the concerns and have also presented the goal of using them into the development process.

The need of formalizing the concern spaces is observed in [9]. In the same paper, the relation between concern spaces and units is expressed using graphs, and, for this purpose, a tool is provided to visualize the graphs. To use all the concerns in spaces, even when some concerns do not apply to certain *units*, the author is proposing to map the units using a default *null*. Also, the author debates if the relation between concerns and units, should be with no restrictions, should be an injective relation or a surjective relation (for each, the *units space* is the function domain).

SoC is used in terms of organizational concepts, instead of programming concepts in [1]. Using this approach, the authors are focusing on closing the semantic gap between a software system and their operational environment. *Tropos*, the methodology they use in software development, is based on modelling early requirements using concepts. Tropos comes with five concern spaces *classes*: actors, resources, hard goals, soft goals and tasks.

The software concerns are modeled in *Cosmos* [16]. The primitives of this approach are concerns, relationships and predicates. There are two large concern categories in Cosmos: Logical and Physical. The logical ones are used to describe concepts related to a system or artefact (examples, issues, features, properties), while the physical ones are pointing to entities of the system or to software artefacts that can be related to logical concerns. There are five logical concern categories: clasifications, classes, instances, properties and topics used to express concerns like functionality, behaviour, performance, robustness, state, coupling, configurability, usability, size, and cost. The "real world" entities of a system: hardware, software, subsystems, services are part of the Physical Concerns. They are divided in three categories in Cosmos: instances, collections (as collections of instances or collections of subcollections) and attributes (attributes of instances or collections).

The ModelSoC approach [8] applies SoC to all artefacts, referring initially to documents, models and code. They use their own *hyperspace* model for a multi-dimensional SoC in order to trace data reused in other models. The *Reuseware* framework has been provided as a tool to relate the information available during the development process to a concern space and generate different views and a final version of the system. The multiple dimensions are applied here to artefacts, and not to concerns.

In another approach [14], *hyperspaces* are used to create a SoC on multiple levels. The *units* are separated on means of different facets of a concern,

providing support to an efficient remodularization processes. As envisioned in this approach, concerns are flexible, being capable of interactions and overlappings. Still, not all entities can be related to hyperspaces, limiting the approach to just some entities.

Another paradigm [17], emphasizes the role of the software artefacts and supports the modelling and implementation process of the artefacts. The approach separates cross-cutting concerns in multiple dimensions by using operations of composition and decomposition. The approach is extended in [6] and is part in all stages of software development, providing support for flexible and traceable artefacts.

Requirements are modeled by using SoC in [2]. The application model is transformed, being divided into chunks. When the transformation of the model is completed, the SoC is used to model and transform requirements in an iterative, complete manner and link them to the application.

The Multi Dimensional SoC approach [12] is used in dealing with requirements. The traditional approach of representing requirements using viewpoints, use-cases is dropped in favor to a conceptualized approach that treats equally functional and non-functional requirements. The requirements space is viewed from two logical perspectives, the system space and the meta-concern space. The System Space consists of all existing requirements, so when discussing an application, the application requirements are already included in the System Space. The concerns in the concern space are typical sets of concerns that can be found in various systems. The relation used to map concerns from the Concern Space to requirements of the System Space in an injective function.

The ARCaDE (Aspectual Requirements Composition and Decision Support) approach [11] treats *PREView* concerns as aspects and separates them when dealing with the requirements. The approach supports concern and candidate aspects identification and specification. Also, candidate aspects are composed with the cross-cutting viewpoints.

The *Aspect-Oriented Software Development with Use Cases* approach [7] handles cross-cutting concerns using Use Cases as overlays on top of classes. The technique proposed manages the cross-cutting concerns individually and composes them to obtain the solution system. The approach is influenced by AspectJ and HyperJ vocabulary.

3. MultiCoS Approach

The MultiCoS concept was introduced in [4] and was extended with the approach and the primitives in [5]. MultiCoS is based on the approach presented in [12]. Compared to other methodologies based on aspect oriented or

based on SoC, the current approach offers versatility, applying concerns on any type of entities, logical or physical, to humans and technology involved in the project, to artefacts including, but not limited to project documents, requirements and code. In the following, we will present the primitives of the approach and the mappings that can be established using the primitives. Then, the main contribution of this paper will be presented: a metric to calculate the *similarity* of different entities. The *similarity* will be established by *similarity indices*, which are calculated using mappings performed to the entities over some common Concern Spaces.

The benefits of the proposed approach over the similar existing ones is that this approach, as will be proved next, can be used with any concern related approach, providing a general-purpose concern modelling capability, that brings the possibility to compare approches, using the primitives of the MultiCoS. Being a generic approach, MultiCoS can be applied to a wide range of situations, not related only with software development.

Primitives. The primitives used are: concern, concern space, entity, entities space, concern value, multispace.

- The *Concern Space* is a group of concerns with cohesion, reffering to/ describing similar properties of certain type of entities. The Concern Space sets a range of values to their concerns [0, max_value]. The concern space has to include at least one concern (dimension).
- The *Concern* is one of the dimensions in a concern space. A concern has the goal to describe the attributes of an entity by assigning a corresponding value from the range to the relation. The relation expresses how much the entity is concerned of a descriptor of its attributes, quantifying the importance of the concern to the entity.
- The *Entity* is an object that can be described using a mapping to different concerns in one or more concern spaces. Examples of entities are: requirements, artefacts, users, software modules, technologies, client specifications, even other abstract objects like concern spaces.
- The *Entities Space* is a collection of entities with cohesion, such as the requirements space, viewpoints space, developers space. Sometimes is referred as *System Space*.
- The *Concern Value* describes the strength of the relation between a concern and an entity. The value reflects the weight of a concern to the entitiy, compared to the other dimensions of the same concern space.

Mappings For every entity, there is a vector from each Concern Space that is associated with the entity, expressing the relation of the entity to the Concern Space. This association can be expressed as a function with the

entity and the concern space as the arguments, and with the associated vector of concerns as a result. Let e be an entity in the system space E, S a concern space with n dimensions and f a mapping function that describes the relation between the entity e and the dimensions in the concern space S. Given this, we can write that:

$$f(e, S) = v, \tag{1}$$

where $e \in E$, $v \in R^n$, and v is the vector with the position coordinates of the entity e in the concern space S.

The concern space S is described using the following format:

$S = (id, name, description, list\ of\ concern\ dimensions, max\_value)$

The id is a unique identification label, $name$ is the unique name of the concern space, the $list\ of\ concern\ dimensions$ is the list of the n concerns of the space. An entity can be assigned a value for a dimension in the space in the range [0, max_value]. max_value is the maximum value an entity can get for any dimension in the space, and is a positive, non-zero value.

In [5] was demonstrated that the MultiCoS approach can be used to represent the concern-entity relation in other methodologies based on separation of concerns. The unitary notation provided can replace the primitives used in other investigated methodologies, which makes easier to compare the methodologies. Also meaning is added to the process represented and can be related to a wide range of meta systems.

$ModelSoC$ approach [8] is extended from the hyperspace model introduced in [14]. The approach is applied on Model-Driven Software Development (MDSD), where the models of the system are the artefacts presented in different formats (views). During the process, the model is composed using DSMLs (Domain-Specific Modelling Languages) and delivered to five different viewpoints (such as diagrams, documents, code) by 12 different composition systems (controllers). These composition systems are: (usecase, participation, exchange, flow, trigger, factory, class, dataclass, associate, typebind, app, security [8]. The relation between viewpoints and composition systems, can be expressed in MultiCoS terms, using the compositioon systems as concerns and the viewpoints as entities.

S_0 = (0, Composition Systems, Composition Systems of concerns, (usecase, participation, exchange, flow, trigger, factory, class, dataclass, associate, typebind, app, security), 1)

The mapping function f using entities from the ViewPoint System Space as the first argument and the Concern Space S_0 as the second will give us the following *coordinates* for the entities in the S_0 concern space:

$f(OpenOffice, S_0) = (1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$

$f(UML\ use\ case, S_0) = (1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0)$

$f(Value\ Flow, S_0) = (1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0)$
$f(UML\ class, S_0) = (0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0)$
$f(Java, S_0) = (1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$

We can notice that for each entity above, in the mapping vector there is more than one non-zero value, meaning that the concerns with non-zero values impact the entity on a certain degree. Concerns which impact different entities are potentially cross-cutting concerns in relation with the entities.

Metrics We can consider that entities are *similar*, if the f function gives *close* vectors for entities in concern spaces. A metric will be used to measure the similarity of two entities.

Let there be two entities a, b in the entity space E, $a, b \in E$, and a concern space S_1, with n dimensions. In first place we will measure the similarity of the two entites, a and b, regarding the value of the f function for the two entities in the concern space S_1.

For each dimension i in the concern space S_1, we will calculate ds_i, the *dimension similarity coefficient*, based on three values, two values are a_i and b_i, the entities' corresponding values in dimension d_i and the max_value set for the concern space S_1.

$$ds_i = 1 - \frac{|a_i - b_i|}{max\_value} \tag{2}$$

Given that, for two null values, $a_i = b_i = 0$, we get $d_i = 1$, meaning *maximum similarity*, we will exclude such *dimension similarity coefficients* from further calculations, taking into consideration that the lack of a particular property for two entities can not lead to a strong *similarity*. Let us consider to have p_1 *dimension similarity coefficients* for which a_i, b_i are not both null values, $p_1 \leqslant n_1$. These p_1 dimension similarity coefficients will be considered *valid*.

The *space similarity coefficient ss*, of two entities a, b over a concern space S_1 is the average of the p_1 *valid* similarity coefficients out of the n_1 dimensions in the concern space S_1.

$$ss_1 = \frac{\sum_{e=1}^{p_1} ds_e}{p_1} \tag{3}$$

The *multispace similarity coefficient*, ms, of two entities a, b over a set of k concern spaces $S_1, ..., S_k$ is the average of the space similarity coefficients of all k spaces in the concern spaces selection with respect to the number of

dimensions.

$$ms = \frac{\sum_{j=1}^{k} ss_j}{k} \tag{4}$$

All these coefficients: dimension similarity (2), space similarity (3), multi-space similarity (4), are taking subunitary values in the range of [0,1], where 0(zero) represents the weakest similarity for two entities (no similarities), while 1 represents the strongest similarity (identical entity concerns).

To conclude this section, the main advantage of the MultiCoS approach is tracing entities with *similar* concerns, and the ability to search entities that are matching specific concern profiles in a certain measureable level.

4. Case study MultiCoS

A Case Study using the MultiCoS approach was conducted. The scenario is that a web application had to be developed. Beside the regular resources provided in order to develop the product, a web application already developed is available with all its artefacts marked up using the MultiCoS approach. This means the existing web application has at least three system spaces (Requirements Space, Documents Space, Code Space) connected to each other by conceptual bindings, based on their relations to the Concern Spaces in the application. The goal is to reuse documents and code from the existing web application to support the development of the new one.

Being established this, we will consider two different web applications, Application A, already developed, called *Read-eng*, an ecommerce web-application selling books, and, Application B, required to be developed, *Target-Ear*, a multimedia web-application for listening music online.

In the Application A, Read-eng, the user can search and view books in store, he can add/remove books in the shopping cart, and he can start the process of buying the books if he is logged in. In the Application B, Target-Ear, the user can search and listen to individual song, he can add/remove song in a playlist, and he can play the playlist only if he is logged in.

The applications are quite different, from services provided, to the potential customers. Still, there are certain functionalities we can relate in the two applications. So, users in both applications can search for content, can open/view an *item* (book/song), can manage a list of items, and they both can use the items in the list for the intended purpose in case they are logged.

In the following, the functionalities of the two applications are written as requirements. The requirements in each application will form a separate Requirements Spaces. Using their value of the f function in the concern spaces, we will relate the requirements in the two applications, establishing

conceptual bindings between certain requirements. When such relations are being established, using a similar process, relations can be created between the entities from the Document Space and from the Code Space in the two applications. The way the requirements are set in the Table 1 gives the reader the impression that requirements are similar. We will check if requirements are similar using MultiCoS aproach.

TABLE 1. The Requirements Spaces of the applications A and B

Aplication A Requirements Space	
id	**Requirement**
RA1	Search book
RA2	View book
RA3	**Add book to shopping cart**
RA4	Remove book from shopping cart
RA5	Login
RA6	Logout
RA7	Create account
RA8	**Logged user can buy items in shopping cart**

Application B Req. Sp.	
id	**Requirement**
RB1	Search song
RB2	Listen song
RB3	**Add song to playlist**
RB4	Remove song from playlist
RB5	Login
RB6	Logout
RB7	Create account
RB8	**Logged user can listen songs in playlist**

Some concern spaces considered for this case study are meant to describe standard properties of different entity types in any type of software systems, others are specific to web applications.

In the last column of Table 2 we notice that Concern Spaces can be included in multispaces collections based on the type of entities they can concern. Concern Spaces 1, 2, 3, 4, 9, 10, 11 can be applied on entities of r type ($r = requirements$), Concern Spaces 1, 2, 4, 5, 7, 8, 10 can be applied on entities of c type ($c = code$).

Knowing this, two requirements, or two snippets of code can be searched for being *similar*. Considering requirement RA8 from application A and requirement RB8 from application B, we will establish the value of f function for each one of them in the concern spaces and we will calculate the space similarity coefficient for each space and the multispace similarity coefficient.

As we can see, in Table 3, the multispace similarity coefficient for RA8 and RB8 is not very high, just 0.627. This happens due to low *space similarity coefficients* on concern spaces S9, S10, S11, S4, as the two requirements do concern different activities (buy vs. play), they are involved differently in the MVC context, they are linked differently to the NFR, and they involve different actions to the DB. Still, given the high values of $ss1$ and $ss2$, some artefacts used in the process are *similar* and can be reused.

A different kind of situation occurs considering the *code* that implements RA3 and RB3.

TABLE 2. Samples of Concern Spaces used in MultiCoS

id	Concern Space:m*	Concern Dimensions	for
S1	**Type of user:1**	Admin, superuser,user, guest	r,c**
S2	**Info Source:1**	Database, user, hostmachine, application	r,c
S3	**Prioritization:10**	Value to Customer, Value to Business, Implementation Cost, Ease of Implementation, Time to Implement	r
S4	**DB Action type:1**	Create, Read, Update, Delete, Static	r,c
S5	**Navigation type:1**	Get, Post, Header, Hyperlink, State	c
S6	**App type:10**	Integrated system, Eshop, Multimedia, Ubiquitos, Social	t,h
S7	**Serv. side lang.:1**	Python, Php, ASP, Ruby, Perl, .NET, C#	c
S8	**Data format:1**	DB, text, XML, UML, HTML, CSV, richtext, DOM, xls	c,t,a
S9	**Activity details:1**	add to list, remove from list, view details, buys, plays, logs	r,c,u
S10	**MVC:10**	Model, View, Controller	r,c,t
S11	**NFR:10**	sleekdesign, loadspeed, volatility, security	r,t

*m=max_value

**r=requirements, c=code, t=technology, a=artefacts, u=user, h=architecture

TABLE 3. The multispace similarity coefficient for requirements RA8 and RB8

Concern Space	Requirement RA8	Requirement RB8	Space similarity coefficient
S1	f(RA8,S1) = (0,1,1,0)	f(RB8,S1) = (0,1,1,0)	ss1 = 1
S2	f(RA8,S2) = (0,1,0,1)	f(RB8,S2) = (0,1,0,1)	ss2 = 1
S3	f(RA8,S3) = (5,10,9,9,8)	f(RB8,S3) = (10,9,9,8,9)	ss3 = 0.84
S4	f(RA8,S4) = (1,0,1,0,0)	f(RB8,S4) = (1,1,0,0,0)	ss4 = 0.33
S9	f(RA8,S9) = (0,0,0,1,0,1)	f(RB8,S9) = (0,0,1,0,1,1)	ss9 = 0.25
S10	f(RA8,S10) = (8,0,10)	f(RB8,S10) = (5,8,10)	ss10 = 0.37
S11	f(RA8,S11) = (7,5,2,10)	f(RB8,S11) = (9,10,6,5)	ss11 = 0.60
		multispace similarity	**ms=0.627**

The high *multispace similarity coefficient* obtained in Table 4, indicates that the code implementing requirement RA3 in application A: "*add book to shopping cart*" can be reused to implement the requirement RB3 in application B: "*add song to playlist*". The high value of the coefficient calculated indicates

TABLE 4. The multispace similarity coefficient for code implementing requirements RA3 and RB3

Concern Space	**code for req. RA3**	**code for req. RB3**	**Space similarity coefficient**
S1	f(RA3,S1)=(0,1,1,1)	f(RB3,S1)=(0,1,1,1)	ss1 = 1
S2	f(RA3,S2)=(1,0,0,1)	f(RB3,S2)=(0,1,0,1)	ss2 = 1
S4	f(RA3,S4)=(0,1,1,0,0)	f(RB3,S4)=(0,1,1,0,0)	ss4 = 1
S5	f(RA3,S5)=(0,1,1,0,0)	f(RB3,S5)=(0,1,1,0,0)	ss5 = 1
S7	f(RA3,S7)=(0,1,0,0,0,0,0)	f(RB3,S7)=(0,1,0,0,0,0,0)	ss7 = 1
S8	f(RA3,S8)=(1,0,1,0,1,0,0,0,0)	f(RB3,S8)=(1,0,1,0,1,0,0,1,0)	ss8 = 0.75
S9	f(RA3,S9)=(1,0,0,0,0,0)	f(RB3,S9)=(1,0,0,0,0,0)	ss9 = 1
S10	f(RA3,S10)=(8,5,7)	f(RB3,S10)=(6,5,6)	ss10 = 0.9
		multispace similarity	**ms=0.969**

that the two snippets of code have the same semantic, even they are not identical, and the applications they serve are quite different. Given this, the code of the application A can be reused, with small adjustments, in developing a new one, the application B.

5. Further work and conclusions

The current approach and the metric presented can be extended to calculate similarity coefficients for applications and for different types of systems. A framework under development have to be completed to support this approach. Investigating and determining proper concern spaces can extend the work of Poshyvanyk et all in [13]. One goal of the approach is to add semantic to the entities using Concern Spaces. The process of mapping entities in the concern spaces using the function f is a non-automatic process, performed by a person, based on his/her considerations, and might be different from one person to another one. As the similarity coefficients are calculated based on these mappings they can have slightly different value. The way the coefficients are calculated and the fact that their values are in the [0,1] range can relate the work presented here to fuzzy mathematics.

Acknowledgements: This work was possible with the financial support of the Sectoral Operational Programme for Human Resources Development 2007-2013, co-financed by the European Social Fund, under the project number POSDRU/107/1.5/S/76841 with the title Modern Doctoral Studies: Internationalization and Interdisciplinarity.

References

[1] Castro J., Kolp M., and Mylopoulos J. Towards requirements-driven information systems engineering: the Tropos project - *Information Systems 27* (2002) 365-389.
[2] Chen X.., Liu Z.., Mencl V., Separation of Concerns and Consistent Integration in Requirements Modelling. Macao, China, 2007.
[3] Dijkstra, E. A Discipline of Programming. 0-13-215871-X. Prentice-Hall 1976, pp 15-25.
[4] Gal-Chis C.E.N., A Multi-Dimensional Separation of Concerns of the Web Application Requirements, *Studia Universitatis Babes-Bolyai, Inf.*, V. LVIII, nr. 3, 2013, pp 29-40.
[5] Gal-Chis C.E.N., Modeling Concern Spaces Using Multi Dimensional Separation of Concern *International Journal of Computers and Techology Vol 11, No 2*: IJCT 2013, pp 2302-2313.
[6] William Harrison W., Ossher H., Tarr P. General Composition of Software Artifacts, *Proceedings of Software Composition Workshop 2006*, March 2006, Springer-Verlag, LNCS 4089, pp 194-210.
[7] Jacobson I., Ng P.W., *Aspect-Oriented Software Development with Use Cases*, Add.-W., 2004.
[8] Jendrik J., Uwe A. Concern-Based (de)composition of Model-Driven Software Development Processes, *Model Driven Engineering Languages and Systems* 2010 pp 47-62.
[9] Kaminski P. Applying Multi-dimensional Separation of Concerns to Software Visualization - *Workshop on Advanced Separation of Concerns*, ICSE 2001.
[10] Kiczales, G., J. Lamping, A. Mendhekar, C. Maeda, C. Videira Lopes, J.-M. Loingtier, J. Irwin. Aspect-Oriented Programming, in: *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP 1997)*, Jyvskyl, Finland, Lecture Notes in Computer Science 1241, Springer-Verlag, pp 220-242.
[11] Moreira A., Rashid A., Arajo J., Modularization and Composition of Aspectual Requirements, in *2nd Aspect-Oriented Software Development Conf.*, Boston, USA, 2003.
[12] Moreira A., Rashid A., Arajo J., Multi-Dimensional SoC in RE. *IEEE*, 2005.
[13] Poshyvanyk, D., Gethers, M., and Marcus, A., Concept Location using Formal Concept Analysis and Information Retrieval, *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 21(4), 2012.
[14] Ossher H., Tarr P. , MultiDimensional SoC and the Hyperspace Approach, 2002.
[15] http://trese.cs.utwente.nl/taosad/separation_of_concerns.htm.
[16] Sutton Jr., S. M., Rouvellou, I. Modeling of Software Concerns in Cosmos. *1st International Conference on Aspect-Oriented Software Development*, Enschede, NL, April, 2002.
[17] Tarr, P., H. Ossher, W. Harrison, S.M. Sutton Jr. (1999): N Degrees of Separation: Multi- Dimensional Separation of Concerns, in: *Proceedings of the 21st International Conference on Software Engineering*, Los Angeles, California, USA, IEEE Computer Society Press, pp107-119.

Faculty of Mathematics and Computer Science, Babes-Bolyai University, Cluj-Napoca

E-mail address: calin.gal-chis@ubbcluj.ro