

LATVIJAS UNIVERSITĀTES  
RAKSTI

751. SĒJUMS

Datorzinātne un  
informācijas tehnoloģijas

SCIENTIFIC PAPERS  
UNIVERSITY OF LATVIA

VOLUME 751

Computer Science and  
Information Technologies

SCIENTIFIC PAPERS  
UNIVERSITY OF LATVIA  
VOLUME 751

# Computer Science and Information Technologies

LATVIJAS UNIVERSITĀTES  
RAKSTI

751. SĒJUMS

# Datorzinātne un informācijas tehnoloģijas

UDK 004(082)

Da 814

Editor-in-Chief:

Prof. **Jānis Bārzdīņš**, University of Latvia, Latvia

Deputy Editors-in-Chief:

Prof. **Rūsiņš-Mārtiņš Freivalds**, University of Latvia, Latvia

Prof. **Jānis Bičevskis**, University of Latvia, Latvia

Members:

Asoc. Prof. **Andris Ambainis**, University of Latvia, Latvia

Prof. **Mikhail Auguston**, Naval Postgraduate School, USA

Prof. **Guntis Bārzdīņš**, University of Latvia, Latvia

Prof. **Juris Borzovs**, University of Latvia, Latvia

Prof. **Janis Bubenko**, Royal Institute of Technology, Sweden

Prof. **Albertas Caplinskis**, Institute of Mathematics and Informatics, Lithuania

Prof. **Jānis Grundspenķis**, Riga Technical University, Latvia

Prof. **Hele-Mai Haav**, Tallinn University of Technology, Estonia

Prof. **Kazuo Iwama**, Kyoto University, Japan

Prof. **Ahto Kalja**, Tallinn University of Technology, Estonia

Prof. **Audris Kalniņš**, University of Latvia, Latvia

Prof. **Jaan Penjam**, Tallinn University of Technology, Estonia

Prof. **Kārlis Podnieks**, University of Latvia, Latvia

Prof. **Māris Treimanis**, University of Latvia, Latvia

Prof. **Olegas Vasilecas**, Vilnius Gediminas Technical University, Lithuania

Scientific secretary:

**Lelde Lāce**, University of Latvia, Latvia

Editor: **Māra Anteniške**

Layout: **Ieva Tiltiņa**

Visi krājumā ievietotie raksti ir recenzēti.

Pārpublicēšanas gadījumā nepieciešama Latvijas Universitātes atļauja.

Citējot atsauce uz izdevumu obligāta.

All the papers published in the present volume have been reviewed.

No part on the volume may be reproduced in any form without the written permission of the publisher.

ISSN 1407-2157

ISBN 978-9984-45-119-0

© Latvijas Universitāte, 2009

## Contents

### SYSTEM MODELING

*Agnis Stibe, Janis Bicevskis*

Web Site Modeling and Prototyping Based on a Domain-Specific Language 7

*Darius Jurkevicius, Olegas Vasilecas*

Formal Concept Analysis for Concept Collecting and Their Analysis 22

*Algirdas Laukaitis, Olegas Vasilecas*

Automatic Verification of the Conceptual Model and Its Documentation 40

*Justas Trinkunas, Olegas Vasilecas*

Ontology Transformation: from Requirements to Conceptual Model 52

### SOFTWARE TESTING

*Guntis Arnicans, Vineta Arnicane*

Using the Sponsor-User-Programmer Model to Improve the Testing Process 65

*Vineta Arnicane*

Complexity of Equivalence Class and Boundary Value Testing Methods 80



## Web Site Modeling and Prototyping Based on a Domain-Specific Language

**Agnis Stibe, Janis Bicevskis**

University of Latvia, 19 Raiņa Blvd, LV-1459, Rīga, Latvia  
*Agnis.Stibe@gmail.com, Janis.Bicevskis@lu.lv*

In the history of software development, there is a haven of different methodologies, approaches, and tools that have been designed to capture the expectations of customers and transfer them into requirements for information systems understandable by programmers. However, reality shows that the percentage of failures in development of information systems in terms of time, money, and functionality is not decreasing.

This paper describes creation of a principle for and development of a domain-specific web site modeling language in order to make interactive dialogue during the very beginning of web site development process to match the expectations of both the customer and supplier. We have developed a tool to implement all the possibilities offered by language. The web site modeling tool has a corresponding component to each general function of the language. The web site model can be simulated within the tool and serve as a prototype for the desired web site in reality. Prototyping approach eliminates barriers between business and information technology people, leading to common understanding of web site goals and success in delivery and implementation. For a more practical view on the issue, the example of the State Revenue Service of the Republic of Latvia web site is examined in this paper.

**Keywords:** software engineering, modeling, specification languages, domain-specific languages, web requirements.

### 1 Introduction: the Variety of Requirements' Specifications

One of the most marked problems in development of any kind of information systems (IS), including web site development, is gathering and consolidation of adequate requirements. Professionally collected, they determine not only the functionality of the developed web site, but also serve as requirements for testing at the phase of acceptance. The requirements' analysis is considered to be a key step in the development of successful IS by all software engineering approaches [1]. Empirical data demonstrate that efforts invested in a detailed requirement analysis considerably reduce drawbacks in later phases of the development [2].

One of the following approaches is most frequently applied in development of requirements.

- Requirements are formulated by business-oriented people, who are less experienced in information technology (IT) issues. As a result, requirements are more informal, incomplete, and sometimes even inconsistent, leaving them to the imagination and interpretation of IT specialists and programmers. That leads to many unexpected changes right after the customer starts to use the newly developed web site.
- Requirements are formulated by IT specialists, which commonly have poorer knowledge of the web site's role and business processes. As a result, there are very well defined requirements that are understandable to web site developers, but may appear to be contradictory to the needs the web site is being produced for.

In practice the solution is quite often based upon the attempts to unite both business-oriented people and IT specialists into a team. Then again, the important question is the choice of the requirement specification language. If requirements are written in a formalized language with explicit semantics, later execution of that specification certainly helps to escape misinterpretation of requirements provided earlier. Unfortunately, Unified Modeling Language (UML) [3] that is currently often recommended is understandable to IT specialists but is less acceptable for business-oriented people. Therefore, UML usage in practice is limited and specifications quite often are written in a natural language, which, in turn, causes ambiguity, inaccuracy, and disagreement about consistency between the specification and the developed system. Common understanding of the planned system – unified communication language – is especially important at the very beginning of system development, when requirements and proposed solutions need to be understood equally by both sides. In many cases the solution is using combined specialized Business Process Modeling Languages (BPML) [4] and workflows. Nevertheless, for the development of web sites (portals, home pages, etc), business modeling languages are less appropriate.

In this paper, a simple and practical reasoning-based web site specification language is proposed. It could, at least partially, solve communication problems between business oriented people and IT specialists.

## **2 Statistics: an Example of Problem Segments in Web Site Development**

Below we present an example which will highlight the main ideas of web site specification. A project of national importance – the project of development and maintenance of the portal of the State Revenue Service (SRS) of the Republic of Latvia [5] – is chosen as an example. It is characterized by:

- quality specification, because the developer of the portal was selected by public procurement, where the procurement subject must be clearly defined. Additionally, there was a previous version of the portal in operation, and that helped to formulate requirements for the new portal more precisely;
- a competent developer team with previous experience in development of several portals of national importance.



The statistics about the development and maintenance of the portal were collected from the first quarter of 2005 to the third quarter of year 2008. The acceptance testing of the portal was performed by the customer during first and second quarter of year 2005. The portal was launched and maintenance began in the second quarter of 2005. All events during testing and maintenance of the portal, further “problem events”, were recorded by the customer with the help of a computerized problem recording system. Consequently, the statistics represent only the customer’s view on the project and do not include the developer’s internal communication. It is remarkable that these statistics are complete, meaning that the developer only processed problem notifications officially recorded by the customer, and no other way of communication regarding drawbacks of the portal was accepted. That established reliable ground for completeness and credibility of the collected data.

234 problem notifications were recorded altogether. They can be divided into five general groups represented in Figure 1.

- Errors (55). This group includes only those problem notifications which are obviously interpretable as inconsistency between the operation of the portal and the specification.
- Misinterpreted requests (32). A programmer has developed an application that he or she thinks is consistent with the specification. However, the customer finds an inconsistency with the specification during the phase of acceptance-testing, and the programmer has to admit his or her fault.
- Loose requests (25). Problem notification is recorded, but it cannot be treated as a mistake of a programmer because the formulation in the specification is ambiguous.
- Changed requests (83). The customer changes initial requirements during acceptance-testing and maintenance of the portal, which results into changes within the application.
- New requests (39). The customer formulates new requirements that were not stated in the beginning.

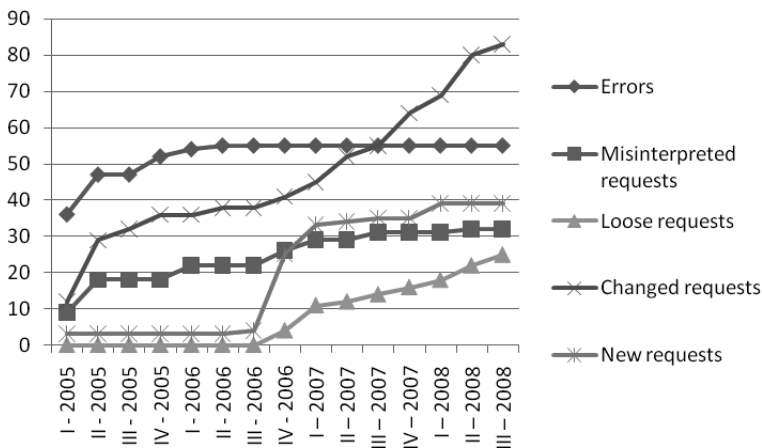


Fig. 1. Problem notifications (cumulative)

The statistics show that mistakes of programmers comprise only 23% of all problem notifications, which lets us question the common belief about the inevitability of programmers' faults. Especially surprising is dispersion of problem notifications in time, which shows that programming errors are solved relatively fast, but the amount of changed requests and additions during the operation of the portal remains stationary high. It is also supported by the fact that 52% of problem notifications correspond to changed requests and new requests; in addition, their amount does not decrease during the maintenance. As a result, the developed but at the same time informal specification was overwritten many times. Actually, there is no unified specification any more, except the one proposed in this paper. The experience of the SRS project demonstrates that similarly to workflow languages that offer specification options to describe operation of information systems, there is a necessity for a particular specification language for the development of web sites, portals, etc, which would let describe easily the main elements of the web site, retaining close relation with the web site operated in reality during its whole lifecycle.

The previously analyzed statistics on the basis of a particular example clearly highlight the following statement: in projects that include development and maintenance of the web site for several years, the main problem is not correction of programmers' mistakes but changed requests in the existing specifications and new requirements. Therefore, a solid ground is established for development of a domain-specific language (DSL) for description of web site elements and their functions.

### **3 The Research: Domain-Specific Language for Web Site Modeling**

In all branches of science and engineering, generic and specific approaches can be distinguished. A generic approach provides a general solution for many problems in a certain area. A specific approach provides a better solution for a smaller set of problems. Both approaches are considered in computer science in relation to the topic "domain-specific languages versus generic programming languages" [6]. A domain-specific language indicates a specification language dedicated to a particular problem domain – web site development in this paper.

On the one hand, currently more efforts are aimed at evolving the existing Semantic Web [7] concepts defined by the World Wide Web Consortium [8] (Resource Description Framework – RDF [9] and Web Ontology Language – OWL [10]) and Universal Networking Language – UNL [11] specifications. Compatible with these concepts is Web Site Parse Template [12], which is a specification for web site structure and content description for web crawlers. It is an effective way to provide web crawlers with proper web page templates to parse web site content more accurately, coordinating the same object attributes used in different pages of the same web site. The Web Ontology Language (OWL) is designed for use by applications that need to process the content of information instead of just presenting information to humans.

For many years, researchers put their efforts in the Semantic Web Service area, work toward further standardization in the area of Semantic Web Service languages and a common architecture and platform for Semantic Web Services [13]. The Web Service Modeling Ontology (WSMO) [14] provides a conceptual framework and a formal language for semantic description of all relevant aspects of web services in

order to facilitate the automation of discovering, combining, and executing electronic services over the Web. Web Service Modeling eXecution environment (WSMX) [15] is the reference implementation of WSMO. It is an execution environment for business application integration where enhanced web services are integrated for various business applications. WSMX internal language is Web Service Modeling Language (WSML) [16]. WSML is based on different logical formalisms, namely Description Logics, First-Order Logic, and Logic Programming, which are useful for the modeling of Semantic Web services. Semantic Web is more perceived as a vision for the future of the Web, in which information is given explicit meaning, making it easier for machines to automatically process and integrate information available on the Web.

Another approach is Web Modeling Language (WebML) [17], a notation for specifying complex web sites at a conceptual level. WebML enables high-level description of a web site by distinct orthogonal dimensions: its data content (structural model), the pages that compose it (composition model), the topology of links between pages (navigation model), the layout and graphic requirements for page rendering (presentation model), and the customization features for one-to-one content delivery (personalization model). All the concepts of WebML are associated with a graphic notation and a textual XML syntax. WebML specifications are independent of both the client-side language used for delivering the application to users, and of the server-side platform used to bind data to pages, but they can be effectively used to produce a site implementation in a specific technological setting. WebML guarantees a model-driven approach to web site development.

Significant characteristics that show why WebML is not suitable for the aim of this paper is that WebML: is a high-level specification language for designing data-intensive web applications; stresses the definition of orthogonal navigation and composition primitives, which the designer can arbitrarily compose to model complex requirements; includes an explicit notion of site view, whereby the same information can be structured in different ways to meet the interests of different user groups or to obtain a granularity optimized for users approaching the site with different access devices; covers advanced aspects of web site modeling, including presentation, user modeling, and personalization.

The UML-based Web Engineering (UWE) [18] approach provides a set of web domain-specific model elements for modeling different concerns describing a web system, such as content, hypertext structure, presentation, and processes. These model elements and the relationships between them are specified by a metamodel. UWE's notation is defined as a lightweight extension of the UML providing the so-called UML Profile for the Web domain. The main focus of the UWE approach is to provide a UML-based domain-specific modeling language, model-driven methodology, tool support for systematic design, and tool support for (semi-)automatic generation of Web applications.

On the other hand, approaches mentioned before are far too sophisticated when the need is to gather, store, and validate customers' requirements for a web site. The solution for any particular problem is going to be discovered and explored in the phase of requirement specification during the process of deeper acquaintance between the customer and supplier. A customer has to be able to define the web site's processes in a simple and precise manner (Web Site Model), at the same time not going deep into programming details.

In turn, a supplier designs a web site prototype and demonstrates it to the customer, who has a chance to evaluate and assess the supplier's proposed solution and its compliance to initial requirements at the very beginning of the web site development.

The approach presented in this paper aims to build an effective environment for collecting, storing, and modeling requirements for web site development, and to make a prototype web site based on these requirements afterwards, allowing everyone to ascertain that the result will meet the expectations. The development of a domain-specific language typically involves a sequence of steps starting with identification of the problem domain, gathering of all relevant knowledge in this domain, and ending with design and implementation of a compiler that translates DSL programs.

In many cases DSL serves as a basis for the development of operable prototypes, which in turn are very good means of mutually harmonized communication for requirement gathering and specification of functionality of developed systems. A deeper insight into prototyping is considered to be a natural extension of the research work, but this paper focuses on the definition of a domain-specific language designed for modeling of web sites, and does not aim to describe the syntax and semantics of the language at this stage.

#### **4 Contribution: Web Site Modeling Language (WeSiMoLa)**

Reviewing the history of recent tendencies in requirement specification, two most common ways can be singled out: one way is high-level specification, where general requirements are specified with no intention to go into the smallest details and functions; the aim is to prepare specification as far as it is understandable for developers in terms of the expected result. The other way is deep programming, where specification includes every detail of the planned system as far as describing each smallest action and reaction to it. In this paper, the first alternative is chosen, because web site environment is developing and changing so fast that it is more reasonable to make high level specification in much shorter time.

The following solution is presented to the problem described in this paper:

- firstly, a web site modeling language, called WeSiMoLa, is created. Additionally, all terms that are simple enough but at the same time satisfactory to formulate web site requirement definition are defined;
- secondly, realization (interpretation) of WeSiMoLa is offered in the form of a prototype, which can be demonstrated to the customer as the web site's prototype.

There are several advantages to the proposed approach. It allows to create an efficient dialog between the customer and supplier in the early stage of requirement specification, enabling to escape the risks of wasted resources (time, money, or others) during elimination of misinterpreted expectations set in inaccurate specifications. The approach assures that the customer always will have specification of their web site that is in compliance with the real up-and-running web site.

WeSiMoLa is a domain-specific language aimed to describe web site's layouts, content, modules, and information structure in an easy readable and constructible manner. WeSiMoLa allows to record web site's content and navigation net in an understandable form that is convenient for other potential users.

The general concept of WeSiMoLa is built upon the idea of the FrameSet approach [19]. The fundamental elements of the language are simple Objects common on the Web, like text, picture, link, etc. As the next step, those objects are combined in different ways to make Frames. Each Frame is built with a specific purpose, e.g., archive, registration forms, questionnaires, and related topics. Finishing the concept of WeSiMoLa, Frames are going to be allocated in FrameSets. The structure can be defined for each FrameSet in terms of number of rows and columns which divide the screen or general view of the web site into several parts. Each of these parts can be filled with one or more Frames.

#### **4.1 FrameSet Definition**

FrameSet is a structure that divides the total space into number of rows and columns, meaning everything that can be viewed by a web browser (window), including space that is outside the screen, which can be reached by scrolling. Rows and columns mark the areas that will serve for a place where Frames will be allocated. It is a physical and logical division that later is visible on the screen. Visibility is enhanced by graphical design. If not, there are always invisible logical lines that separate groups of Frames on the screen. Each area in the FrameSet, whether it is row or column, has a fixed or proportional height and width. In case of fixed parameters, a row or column will remain at a constant size when browser window is resized or resolution is changed. If height or width of some row or column is defined as proportional, the size of the particular area will change according to the updated resolution or proportionally to the resizing of the browser window. Any area in the FrameSet can hold none, one, or many Frames.

#### **4.2 Frame Definition**

Frame can be allocated somewhere in a FrameSet according to its purpose or size. Similarly to FrameSet areas, there can be two kinds of Frames – with fixed height or width or of flexible size that adapts to the size values of an area where it is placed. Each Frame has its purpose or shape. A Frame with a purpose means that it is built to execute a particular process or to serve a particular need. Most common types of purpose Frames are help, archive, registration forms, questionnaires, etc. All other Frames are defined with an aim to represent particular information in a particular shape, e.g. a picture and a textual description, which are Objects. Frames are built with the help of various Objects, combining them to serve the particular purpose of the chosen Frame. Frame has a parameter that allows to show it in a pop-up window or as an overlaying window as well.

#### **4.3 Object Definition**

Objects are the basic elements of WeSiMoLa. They are different and each of them performs a specific task or represents a unique meaning. There are passive Objects like text and graphical elements without the possibility to interact with them; they are only representative elements. All others are active Objects like links in a text or graphical elements; it is possible to navigate away from it, input data and perform a related process with the help of various input fields and action buttons, etc. The number and type of Objects is predefined in the language. The only task is to develop Frames from the Objects and put the Frames into FrameSets.

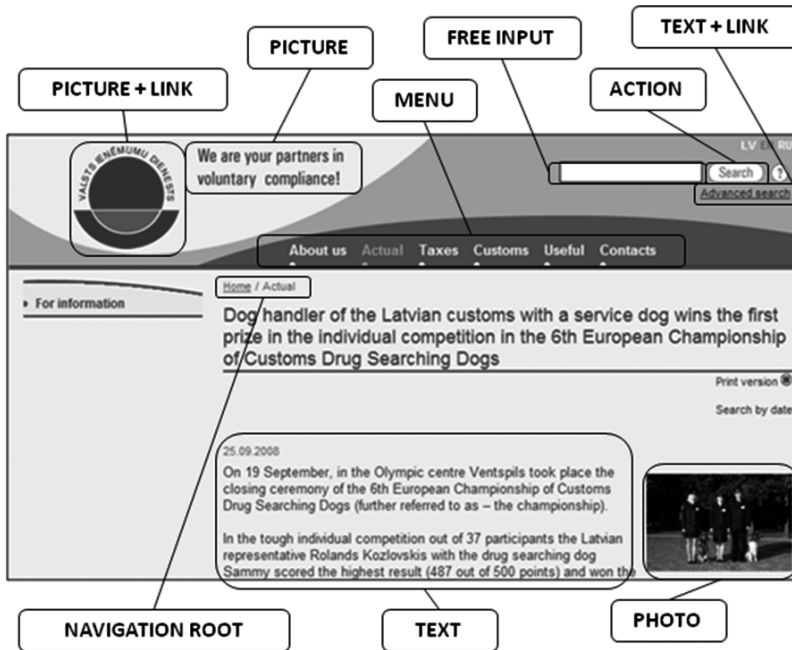


Fig. 2. WeSiMoLa Objects in SRS web site

#### List of predefined Objects:

- Text Element;
- List;
- Table;
- Picture;
- Animation;
- Interaction;
- Link:
  - Internal Frame / Internal Web Site / External,
  - To Page / to File;
- Input Fields:
  - Radio Buttons,
  - Check Boxes,
  - Drop Down,
  - Free Input;
- Action on Input Data;
- Menu:
  - Horizontal / vertical,
  - Levels overlapping horizontally / vertically,
  - Levels expanding vertically,
  - Language;

- Photo / Audio / Video;
- Search;
- Navigation Root;
- Banner;
- Blog;
- Calendar;
- Abstract;
- Login;
- Print this page.

An example of part of WeSiMoLa Objects is represented in Figure 2. For consistency throughout the paper, SRS web site is taken as the example.

## 5 The Web Site Modeling Technique

The basis of the web site modeling technique is composing Frames using predefined Objects, allocating Frames into designed FrameSet templates, and building up the navigation. To support this modeling technique, a Web Site Modeling Tool (WeSiMoTo) is developed. Four general components of web site modeling are:

- information source structured in a tree form for menu purposes and core navigation:
  - in WeSiMoTo this component is embodied in Tree Builder;
- repository for keeping composed Frames:
  - in WeSiMoTo this component is embodied in Frame Composer;
- repository for keeping designed FrameSet layouts:
  - in WeSiMoTo this component is embodied in FrameSet Designer;
- navigation modeling:
  - in WeSiMoTo this component is embodied in Navigation Net.

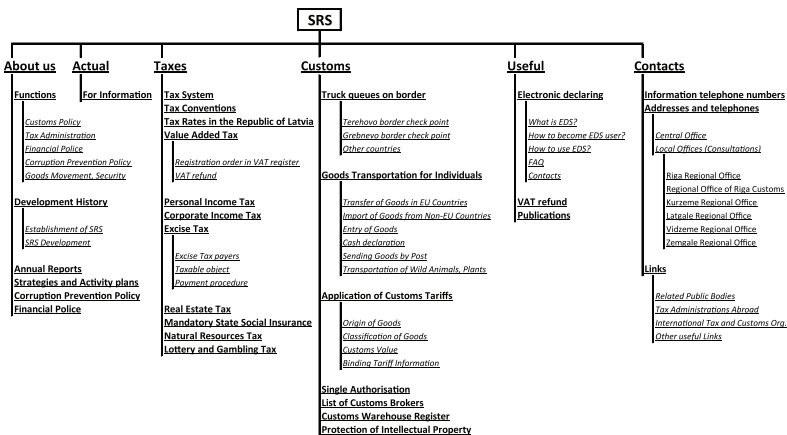


Fig. 3. Full Information Tree of the SRS web site

### 5.1 Information Tree Modeling (Tree Builder)

The information tree represents graphically all the textual information that a web site has and its location in a structured way. It also clearly shows the core navigation through the web site and subordination of information pieces.

Full Information Tree of the English version of SRS web site is represented in Figure 3. It has six general sections and goes deep to the fourth sublevel.

Tree Builder is the component in WeSiMoTo that allows to input and order information in an easy understandable and user friendly environment. Its general functions are:

- create a new tree / save / remove;
- add / remove language of the tree;
- add / edit / remove sublevels of information in the tree;
- add / edit / remove information;
- save tree.

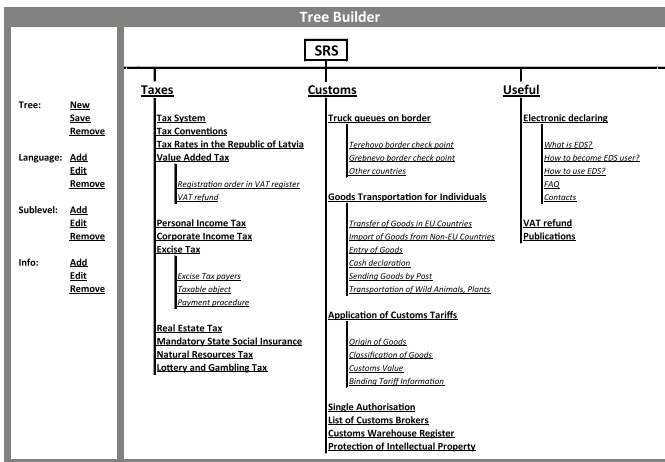


Fig. 4. Tree Builder of the SRS web site

Figure 4 represents a view of the Tree Builder component of WeSiMoTo during the development of Information Tree of the SRS web site.

### 5.2 Frame Modeling (Frame Composer)

Frame modeling gives the opportunity to combine Objects in different ways in order to produce Frames with concrete purposes or with particular shapes. As an example, Figure 5 represents the Advanced Search Frame of the SRS web site. It consists of Text, Picture, Free Input, Drop Down and Action Objects.



Fig. 5. Advanced search Frame of the SRS web site



Frame Composer is the component in WeSiMoTo that supports Frame modeling. It contains all predefined Objects and offers the following functions:

- create new / edit / remove Frame;
- set size parameters for the Frame;
- pop-up / overlaying Frame;
- add / edit / remove Object;
- save Frame in repository.

Figure 6 represents the Advanced Search Frame of the SRS web site previously shown in Figure 5, only here it gives a view of the Frame through the Frame Composer component of WeSiMoTo.

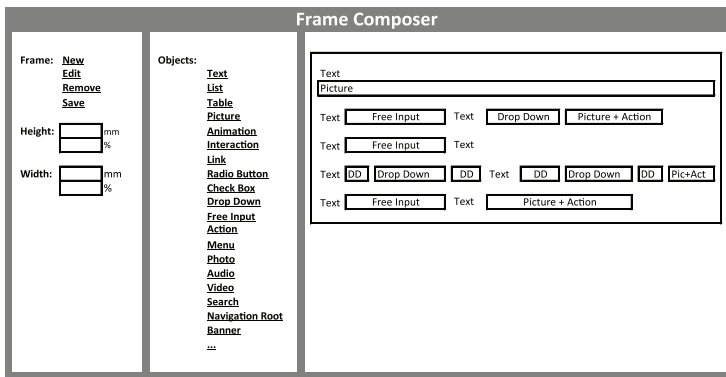


Fig. 6. Advanced search Frame of the SRS web site in Frame Composer

### 5.3 FrameSet Modeling (FrameSet Designer)

The function of FrameSet modeling is to prepare the general layout of a web site; it defines the number of columns and rows in a view, their height and width. FrameSet presents a logical and physical structure indicating where Frames then can be allocated in the browser window.

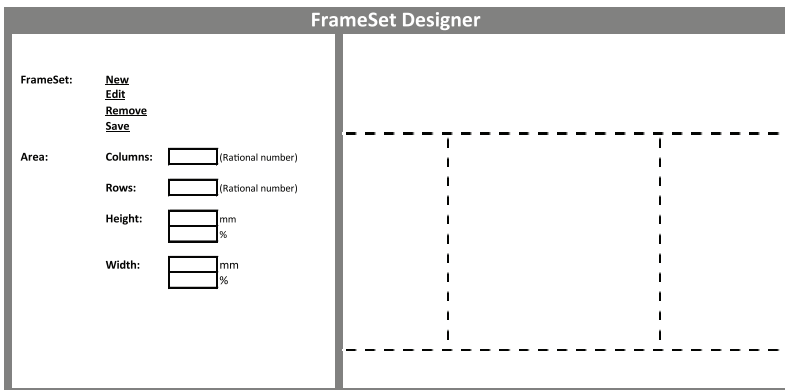


Fig. 7. FrameSet Designer

FrameSet Designer represented in Figure 7 is the component in WeSiMoTo that allows to make as many different FrameSet layouts as needed. Designer offers the following functions:

- create new / edit / remove FrameSet;
- set number of columns / rows;
- set size parameters for each area – fixed or proportional;
- save FrameSet in repository.

#### 5.4 Navigation Modeling (Navigation Net)

Navigation modeling is the essence of web site modeling. It bounds together all parts of web site modeling described before. It allows to construct the net of relations among Information Tree, FrameSets, and Frames, permitting web site navigation.

Navigation Net is the component in WeSiMoTo that takes Information Tree as the basis and allows to choose any prepared FrameSet layout and connect it to a chosen branch or leaf of the tree. Then each FrameSet can be filled with Frames, meaning predefined Frames can be placed into FrameSet areas and connections among them are made. Consequently, navigation is built up.

General functions of Navigation Net are:

- show and navigate Information Tree;
- show and make available all predefined Frames;
- for each tree branch or leaf – attach / show / remove FrameSet;
- for each FrameSet area – add / remove Frame;
- add / edit / remove links from / to Frames and Information Tree;
- save model.

Figure 8 to Figure 11 demonstrate an example of navigation through the SRS web site. Figure 8 is the first in the row where Navigation Net describes the position corresponding to the main page of the SRS web site.

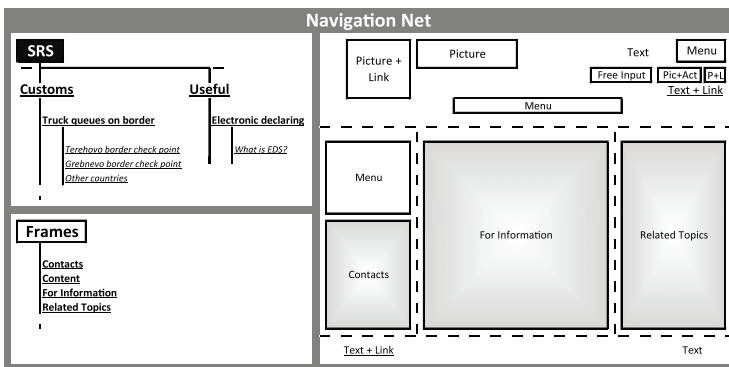


Fig. 8. Navigation Net representing the main page of the SRS web site in English

A Navigation Net window consists of three major parts: Information Tree, Frames, and FrameSet views. Since the space is limited, part of Information Tree can be viewed

and the actual page is marked black. Frames are organized in a tree form and are accessible from Frames view. The right side is left for FrameSet construction, filling them with Frames and building up navigation among them all.

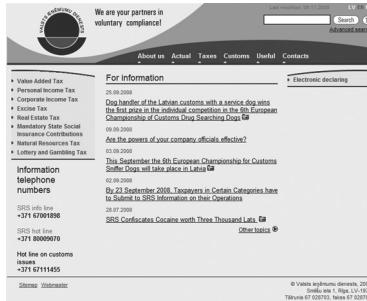


Fig. 9. The main page of the SRS web site in English

Figure 9 represents the same English language main page of the SRS web site, but now in shape of a real site, where all Objects and FrameSet are replaced with actual textual and graphical information.

In order to see how the navigation works, a click on “Related Topics” – “Electronic Declaring” is simulated. Figure 10 represents the actual state after the click. Again, the root to the actual page is marked black, namely “SRS” -> “Useful” -> “Electronic declaring” -> “What is EDS?” Consequently, there are changes in the FrameSet side, where the previous is substituted with the corresponding to the new state in the Tree. FrameSet now has a different set of Frames, their location, and one column less.

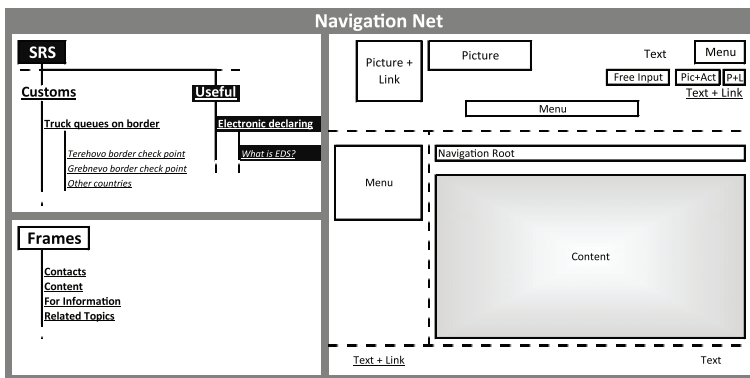


Fig. 10. Navigation Net representing EDS page of the SRS web site in English

Figure 11 represents the same English language EDS page of SRS web site, again in a shape of the real web site, where all Objects and FrameSet are replaced with actual textual and graphical information. This example shows the way the Navigation Net can be used for building up the navigation of the web site and how a prototype is build from the model and later simulated.

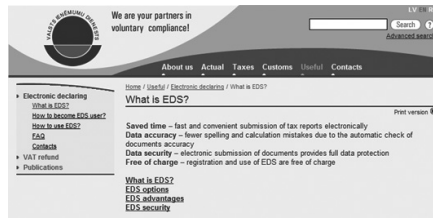


Fig. 11. EDS page of the SRS web site in English

## 5.5 Web Site Model

Web Site Model is a set of internally coherent four types of modeling processes mentioned before. All types of processes, except Navigation Net, and their names are presented in a tree-type structure that shows the content of the model.

The model's content consists of three parts:

- Information Tree's diagram – presents all information in a structured way, and it is unique to the whole model;
- Frames' repository – every row describes one Frame with attached design that shows the content of the Frame – Objects in it and its parameters;
- FrameSets' repository – each row describes one FrameSet with attached layout of the FrameSet and its parameters.

## 6 Conclusions and Further Directions

This paper examined the domain-specific web site modeling language WeSiMoLa and the web site modeling tool WeSiMoTo based on the WeSiMoLa language, which give the possibility to collect, develop and store requirements in an easily understandable way for both customers and suppliers. In contrast to the waterfall principle, the gathering of requirements here is organized on the basis of gradual elaboration approach. Moreover, a web site model can be built from the requirements. A prototype of the web site is the consequent next step that can be executed, thus clearly demonstrating the planned web site in action and assuring both parties of how common their expectations are and whether they will be met. Prototyping is considered as a natural extension of this research paper.

The proposed method suits well the need for acquiring requirement specifications for various kinds of simple type web sites such as HTML-based home pages, promotional web sites, and others of the same sort. The approach described in the paper also solves well the requirement management for web portals based on content management systems. The scope of this research does not include web sites integrated with sophisticated data sources, because those would involve an additional definition of DSL objects for the data storage. That gives an opportunity for further and broader research within the field of web development.

The proposed web site modeling language is built upon practical experience and requirements, therefore it can't be perceived as fixed and complete. Primarily it applies to the types of Objects that are used in the Frame composition process. Evolution of the web will bring ever more sophisticated object types that would need to be included into the specification language.

However, the example of the already existing SRS web site and its model in WeSiMoTo represents how user friendly is the modeling language and how natural is the modeling tool in operation. The advantage of the described language is that in contrast to a general-purpose modeling language such as the UML, WeSiMoLa (with supporting WeSiMoTo) allows a particular type of issues or their solutions on the Web to be expressed more clearly.

In the future this modeling language and the related tool will be investigated more by applying them in different new web site development projects at the requirement analysis stage, thereby achieving greater approbation and higher precision in each of their components. Secondly, this paper does not describe the collaboration between the web site and Content Management System (CMS) and its formalization, which is for a prospect of further research in the future.

## References

1. Lowe D., Eklund J. (2002) Client Needs and the Design Process in Web Projects, *Journal on Web Engineering* 1, Rinton Press, pp. 23–36.
2. Sommerville I., Ransom J. (2005) An empirical study of industrial requirements engineering process assessment and improvement, *ACM TOSEM* 14, pp. 85–117.
3. Unified Modeling Language (UML), version 2.1.2., available online: <http://www.omg.org/technology/documents/formal/uml.htm>.
4. Business Process Modeling Language (BPML), available online: [http://www.service-architecture.com/web-services/articles/business\\_process\\_modeling\\_language\\_bpml.html](http://www.service-architecture.com/web-services/articles/business_process_modeling_language_bpml.html).
5. State Revenue Service of Republic of Latvia, available online: <http://www.vid.gov.lv>.
6. Van Deursen A., Klint P., Visser J. (2000) Domain-Specific Languages: An Annotated Bibliography. Computer Based Learning Unit, University of Leeds, available online: <http://homepages.cwi.nl/~arie/papers/dslbib/>, February 2000.
7. Herman I. (2008) W3C Semantic Web Activity. Available on the internet <http://www.w3.org/2001/sw/>. April 2008.
8. Jacobs I. (2008) About the World Wide Web Consortium (W3C). Available online: <http://www.w3.org/Consortium/>, February 2008.
9. Swartz A. Application/rdf+xml Media Type Registration”, available online: <http://www.ietf.org/rfc/rfc3870.txt>, September 2004.
10. Schreiber G., Dean M., van Harmelen F., Hendler J., Horrocks I., McGuinness D. L., Patel-Schneider P. F., Stein L. A. OWL Web Ontology Language Reference/W3C Recommendation. Available online: <http://www.w3.org/TR/owl-ref/>, February 2004.
11. Universal Networking Digital Language Foundation (UNL). Available online: <http://www.unl.org/>.
12. Manukyan Av., Manukyan Ar., Mailyan A., Sayadyan A. (2008) Website Parse Templates. Available online: <http://tools.ietf.org/html/draft-manukyan-website-parse-templates-00>, April 2008.
13. The mission of the ESSI WSMO working group. Available online: <http://www.wsmo.org>.
14. De Bruijn J., Bussler C., Domingue J., Fensel D., Hepp M., Keller U., Kifer M., Kopecky J., Lausen H., Oren E., Polleres A., Roman D., Scicluna J., Stollberg M. (2005) Web Service Modeling Ontology. Available online: <http://www.w3.org/Submission/WSMO/>, June 2005.
15. Web Service Modelling eXecution environment. Available online: <http://www.wsmx.org/>.
16. De Bruijn J., Fensel D., Keller U., Kifer M., Lausen H., Krummenacher R., Polleres A., Predoiu L. (2005) Web Service Modeling Language. Available online: <http://www.w3.org/Submission/WSML/>, June 2005.
17. Ceri S., Fraternali P., Bongio A. (2000) Web Modeling Language: A Modeling Language for Designing Web Sites. In: Proc. of the 9th Intl World Wide Web Conference, May 2000, Amsterdam, pp. 137–157. Available online: <http://www9.org/w9cdrom/177/177.html>.
18. Koch N., Kraus A. (2002) The Expressive Power of UML-Based Web Engineering. In: Proc. of the Second Intl Workshop on Web-Oriented Software Technology (IWWOST02), Malaga, 2002, pp. 105–119. Available online: <http://www.pst.ifi.lmu.de/projekte/uwe/>.
19. Arnicans G., Karnitis G. (2006) Intelligent Integration of Information from Semi-Structured WEB Data Sources on the Basis of Ontology and Meta Models. In: Proc. of the 7th Intl Baltic Conference DB&IS, Vilnius, 2006, pp. 177–186.

## Formal Concept Analysis for Concept Collecting and Their Analysis\*

Darius Jurkevicius<sup>1</sup> and Olegas Vasilecas<sup>2,3</sup>

<sup>1</sup> Department of Information Systems, Faculty of Fundamental Sciences, Vilnius Gediminas Technical University, Sauletekio al. 11, LT-10223 Vilnius-40, Lithuania, *d.jurkevicius@isl.vgtu.lt*

<sup>2</sup> Information Systems Research Laboratory, Faculty of Fundamental Sciences, Vilnius Gediminas Technical University, Sauletekio al. 11, LT-10223 Vilnius-40, Lithuania, *olegas@isl.vgtu.lt*

<sup>3</sup> Department of Computer Science, Faculty of Natural Sciences, Klaipeda University, Herkaus Manto 84, LT-92294 Klaipeda, *olegas.vasilecas@ik.ku.lt*

The method how to collect concepts and analyse them using formal concepts analysis is presented in this paper. This method allows representing of the hierarchical tree of concepts without analyzing terms in a specific domain. The main idea of the proposed method is to select concepts (objects) from collected data of specific domain using templates. Later the collected terms are analyzed using the formal concept analysis method. This allows to simplify the steps of term analysis and ontology representing in the ontology development process. We propose to use the logical structure of context that allows to extend the traditional context. It allows to save more data in the context and to simplify the usage of context in information systems. We call this extended formal context distributional formal context. In this paper, the real estate domain was selected for the experiment and its results are presented.

**Keywords:** formal concepts analysis, formal context, formal concept, multi-formal context, distributional formal context, concept collecting.

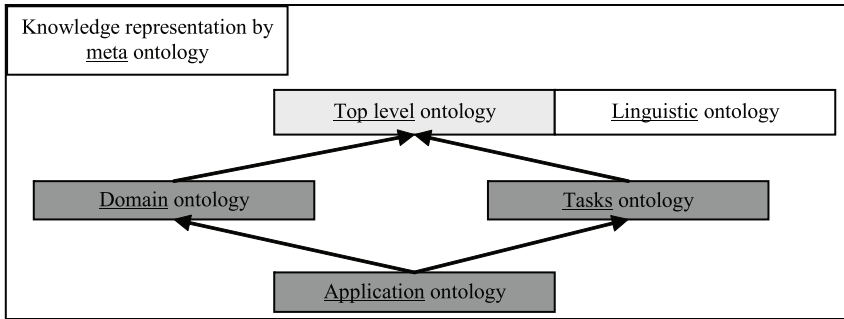
### Introduction

Usually, different types of ontology are used for designing of information systems (Fig. 1). During ontology development process the general concepts (entity, event, date, process, etc) are defined at top level. At the next step, concepts of domain (domain ontology), concepts of process (ontology of process), and concepts of tasks (ontology of problem) are defined. In that way principle of independence of the domain knowledge and knowledge about process proposed by Guarino (5) are implemented. Processes are described by the terms of the actor. The processes in relation with the problem are

---

\* The work is supported by Lithuanian State Science and Studies Foundation according to High Technology Development Program Project "Business Rules Solutions for Information Systems Development (VeTIS)", Reg. No. B-07042.

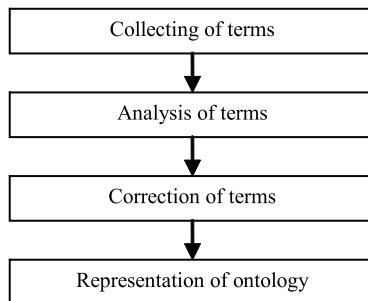
linked to the entity of domain. It is useful to find components that can be repeatedly used for designing information systems. Ontology is one of such components. Ontology of process can be repeatedly used, because its structures are the same in different domain processes. Generally, the ontology development process includes 4 stages: collecting of terms, analysis of terms, correction of terms, and representation of terms. The diagram of this process is shown in Figure 2.



*Fig. 1. Types of ontology*

The identification of all terms of a specific domain, their relationship and definitions are included into the term collection stage. Terms are analysed at the stage of term analysis. During this stage different terms that describe the same objects and processes are searched for. After the searching of terms is finished, the usage of one general term must be achieved. This is done during the third stage of ontology development. The fourth stage is representation of ontology using a specific language (for example, OWL), and during this stage a tool is used. The choice of ontology representation language belongs to the ontology development tool. A user without a specific knowledge can develop ontology just by using ontology development tool. For example, an information system engineer can perform this work.

The ontology development process described above and presented at Fig. 2 is slow and time-consuming.



*Fig. 2. Process of ontology development*

Automating of the stages of term analysis and ontology representation is one of the ways to develop ontology more quickly. There are no systems that could automate these

stages at this moment. However, researchers' community in the ontology development field believes that in achieving the technology of semantic Internet, specific systems will be developed that should automate ontology development stages. This paper proposes the method for automation of ontology developed process by mean of formal concept analysis.

The authors apologize for the readers because some pictures of application are in the Lithuanian language. The application and ontology for it was used for analysing data in Lithuanian. The remaining part of this paper consists of the following sections.

- Formal concept analysis and its usage for ontology developed is described in the second section.
- Collecting of terms in a specific domain and the context logical structure that allows to extend the traditional context is described in the third section.
- The experiment and its results are described in the fourth section. The system architecture is presented.
- The conclusions are drawn in the last section.

## What is Ontology?

According to [1, 14, 15], in the context of computer and information sciences, ontology is defined as a set of representational primitives with which a domain of knowledge or discourse is modeled. The representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members). The definitions of the representational primitives include information about their meaning and constraints on their logically consistent application.

By the feature of expressivity, two types of ontology can be distinguished: heavyweight and lightweight ontology. The main difference between them is the role played by axiomatization. Heavyweight ontologies are extensively axiomatized and lightweight ontologies often are presented as simple taxonomic structures and are either slightly or not axiomatized.

Contemporary ontologies share many structural similarities, regardless of the language in which they are expressed. Most ontologies describe individuals (instances), classes (concepts), attributes, and relations. In this section, each of these components is discussed in turn.

Common components of ontologies can include:

- individuals: instances or objects (the basic or "ground level" objects);
- classes: sets, collections, concepts, types of objects, or kinds of things;
- attributes: aspects, properties, features, characteristics, or parameters that objects (and classes) can have;
- relations: ways in which classes and individuals can be related to each other;
- function terms: complex structures formed from certain relations that can be used in place of an individual term in a statement;
- restrictions: formally stated descriptions of what must be true in order for some assertion to be accepted as an input;



- rules: statements in the form of an “if-then” (antecedent-consequent) sentence that describe the logical inferences that can be drawn from an assertion in a particular form;
- axioms: assertions (including rules) in a logical form that together comprise the overall theory that the ontology describes in its domain of application. This definition differs from that of axioms in generative grammar and formal logic. In these disciplines, axioms include only statements asserted as *a priori* knowledge. As used here, axioms also include the theory derived from axiomatic statements.

## Understanding of Formal Concept Analysis

One of the ways to transform available data in a hierarchic form is formal concept analysis. Dau [2] noticed that scientists making plots could not lean on them as on arguments. To separate formally the mathematical structure from its schematical presentation, the work environment was created in which diagrams could be used to make formal substantiations. Now we will define some terms used in this paper.

**Concept** can be defined as:

- an abstract or general idea inferred or derived from specific instances [9, 23];
- an abstract idea or a mental symbol typically associated with a corresponding representation in language or symbology that denotes all of the objects in a given category or class of entities, interactions, phenomena, or relationships between them [15, 23];
- having an intention (deep definition), extension (set of objects or exemplars) [10];
- the definition of a type of objects or events; a concept has an intentional definition (a generalization that states membership criteria), and an extension (the set of its instances) [11].

**Formal Concept Analysis (FCA)** [17] method is:

- a mathematization of the philosophical understanding of concept;
- a human-centred method to structure and analyze data;
- a method to visualize data and its inherent structures, implications, and dependencies.

FCA is based on the philosophical understanding that a concept can be described by its extension – that is, all the objects that belong to the concept and its intension which is all the attributes that the objects have in common [13], and this can be represented as a table.

**Formal context** is the mathematical structure used to formally describe these tables of crosses (or, briefly, a context) [22].

FCA is a method used in data analysis, knowledge representation, and information control. Rudolf Wille suggested FCA in 1981 [17], and this method is successfully developed nowadays. For the first 10 years, FCA was researched by small groups of scientists and Rudolf Wille’s students in Germany. FCA was not known worldwide

because the bulk of publications were presented at mathematicians' conferences. After getting the sponsorship, some projects were implemented in this area. Most of them were knowledge research projects used for systems development. This system was known only in Germany. During the last 10 years, FCA became the research object of the international scientific community. FCA was used in linguistics, psychology, as well as in software engineering and in the areas of artificial intelligence and information search.

Some of the structures of FCA appear to be fundamental to information representation and were independently discovered by different researchers. For example, Godin et al. [5] used concept lattices (which they call "Galois lattices") for information retrieval.

Now, we shall introduce the definition of formal concept analysis [4]. Let us present an example:  $G$  is the set of objects that we are able to identify in some domain (e.g. if, when, than). Let  $M$  be the set of attributes. We identify the index  $I$  as a binary relationship between the two sets,  $G$  and  $M$ , i.e.  $I \subseteq G \times M$ . The triple  $(G, M, I)$  is called a formal context. For  $A \subseteq G$ , we define

$$A' := \left\{ m \in M \mid (g, m) \in I \text{ for all } g \in A \right\} \quad (1),$$

and dually, for  $B \subseteq M$

$$B' := \left\{ g \in G \mid (g, m) \in I \text{ for all } m \in B \right\} \quad (2).$$

A formal concept of a formal context  $(G, M, I)$  is defined as a pair  $(A, B)$  with  $A \subseteq G$ ,  $B \subseteq M$ ,  $A' \subseteq B$  and  $B' \subseteq A$ . Sets  $A$  and  $B$  are called the extend and intend of the formal concept. The set of all formal concepts of a context  $(G, M, I)$  is called the concept lattice of the context  $(G, M, I)$ .

## The Proposed Method for Concept Collecting and Their Analysis

Our method for term collection and analysis of data is proposed in this section. The sequence of term collection, analysis, and representation is shown in Figure 3. Hereinafter we will describe the process of our proposed method. Firstly, analysts (or users) select the specific domain. The second step is acquisition of information for analyzing the domain. Then the user defines the attributes. Attributes are needed for a specific tasks (for example, criteria attributes are needed for task search). Next, the user inputs all attributes into formal context and the information is analysed (Fig. 5). The user searches for the objects (concepts) in information and, when the object (concept) is found, it is inputted into the formal context and the next step is information processing.

When all objects are inputted into formal context and information processing is executed, the analyst can create the concept net (Fig. 4). Using the created concept net, the analyst can search the dependences, matching, repeated structures, exceptions, etc. The analyst can also develop ontology using the concept net (Fig. 4).

Information system can use the created formal context to solve a specific task.

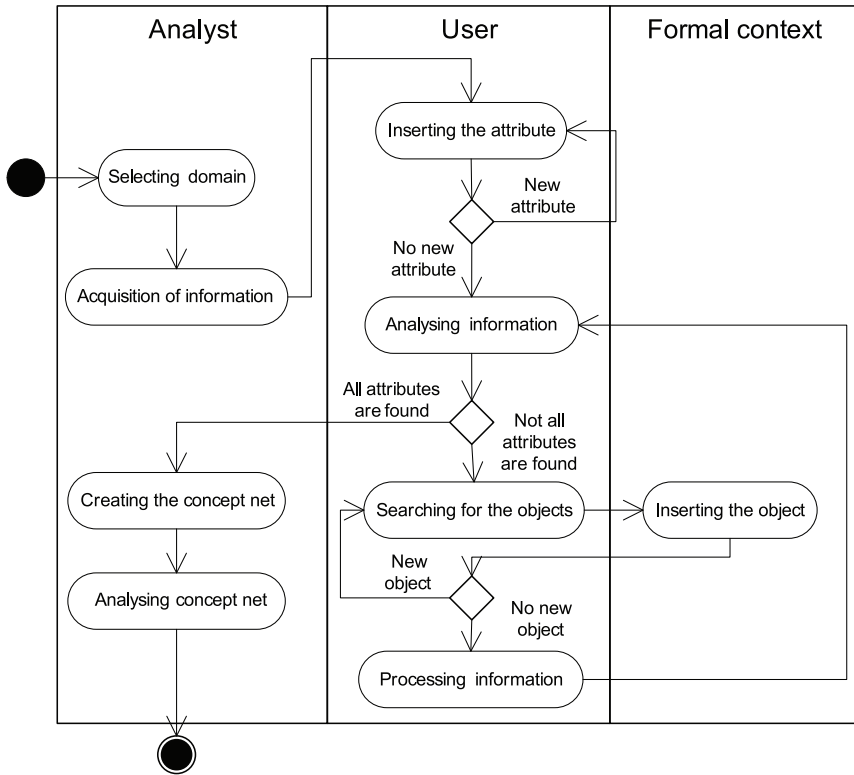


Fig. 3. The process of term collection, analysis, and representation

It is well known that knowledge is important part of modern information systems. Knowledge used in information systems is stored in knowledge bases, data bases, ontology, and other knowledge sources. Formal context is one of knowledge sources.

The traditional formal context is used in formal concept analysis. The traditional context is a triple  $(G, M, I)$  consisting of a set of formal objects  $G$ , a set of formal attributes  $M$ , and a binary relation  $I \subseteq G \times M$  (expressing the attributes pertaining to each object) [3, 18, 19, 20, 21]. The Formal Concept Analysis method is: a mathematization of the philosophical understanding of concept; a human-centred method to structure and analyze data; a method to visualize data and its inherent structures, implications, and dependencies. However, this method is quite difficult in using it for information systems because there is not enough data stored in the context.

In comparison with formal context structure, we review the ontology structure because these structures are quite similar. They have objects (classes), attributes, and relations between objects and attributes. Ontology in computer science and information science is a formal representation of a set of concepts within a domain and the relationships between those concepts. It is used to reason about the properties of that domain, and may be used to define the domain. Most ontologies [7, 12, 16] describe individuals (instances), classes (concepts), attributes, and relations. In this section, each of these components is discussed in turn.

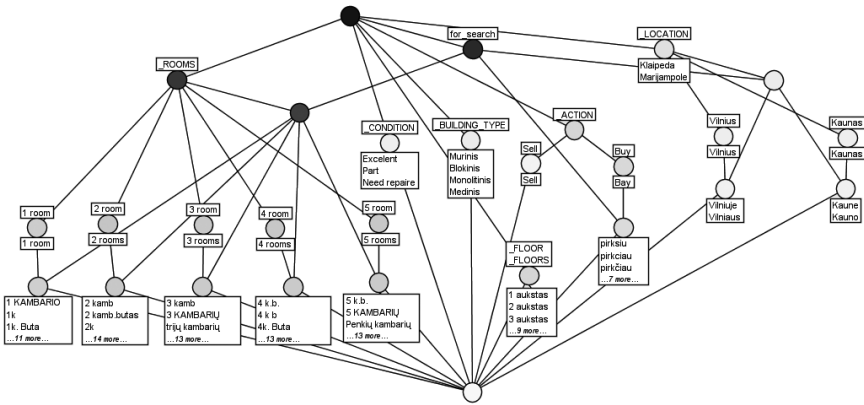


Fig. 4. The net of concepts created with ToscanaJ-1.6 tool from formal context

Ontology structure is more complex than formal context structure.

In the next section, we propose the context logical structure that allows to extend the traditional context. That solution permits saving more data in the context and simplifying the usage of context in information systems.

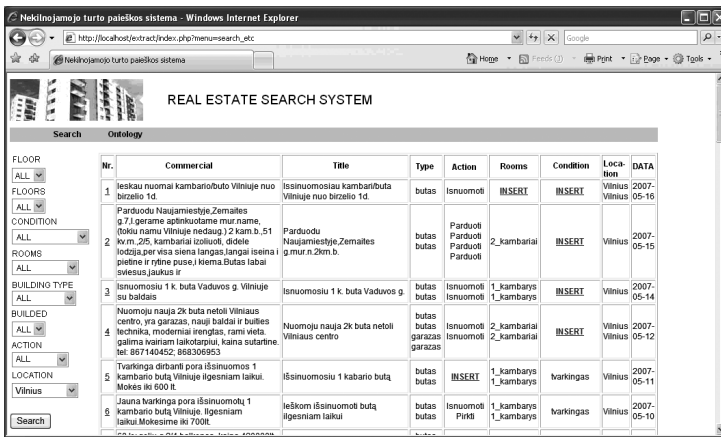


Fig. 5. The form for information analysis

### Distributional Formal Context

A traditional formal context in formal concept analysis is a triple (G, M, I) consisting of a set of formal objects G, a set of formal attributes M, and a binary relation  $I \subseteq G \times M$  (expressing the attributes pertaining to each object) [3, 18, 19, 20, 21]. The traditional logical data model and graphical representation of formal context [4, 22] in formal concept analysis is shown in Figure 1.

For more information storage in formal context, we propose to extend the logical scheme of formal context. That allows keeping general features of formal concept

analysis. By [4, 19], for the mathematical definition of *formal concepts*, we introduce the derivation operators “ ’ ”.

Using the derivation operators, we can derive *formal concepts* from our *traditional formal context* with the following routine:

- 1) pick a set of objects A;
- 2) derive the attributes A';
- 3) derive (A)';
- 4) (A",A') is a *formal concept*.

An example of generating formal concept from traditional formal context (Fig. 6):

- 1) pick any set of objects A, e.g. A = {Object 3};
- 2) derive the attributes A' = {Attribute 3, Attribute 4, Attribute n};
- 3) derive (A)' = {Attribute 3, Attribute 4, Attribute n}' = { Object 3, Object 4};
- 4) (A",A') = ({Object 3, Object 4},{Attribute 3, Attribute 4, Attribute n}) is a *formal concept*.

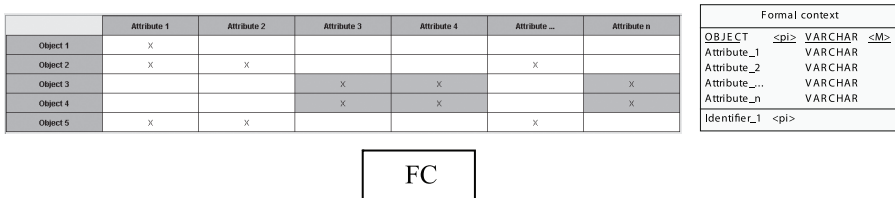


Fig. 6. Traditional formal context (FC) scheme and symbol

We propose to divide the traditional formal context into three parts:

- objects are described in the first table (Objects);
- attributes are described in the second table (Attributes);
- relations between objects and attributes are described in the third table (Relation). The proposed physical data model is shown in Figure 7.

To separate the distributional formal context (DFC) from traditional formal context (FC), we propose graphical notation:

- the traditional formal contexts are represented as rectangles (Fig. 6);
- the distributional formal contexts are represented as rectangles with bias (Fig. 7).

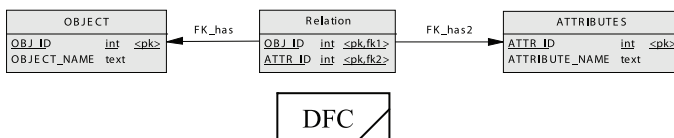


Fig. 7. The proposed logical scheme and notation of distributional formal context (DFC)

However, formal concept analysis needs traditional context (Figure 6). To convert distributional formal context to traditional context, additional transformation is needed. We propose the transformation table where relations between objects and attributes are saved. That transformation is used for formal concept analysis. In Figure 8 relation table transformations into traditional context are shown. The traditional formal context is created after transformation.

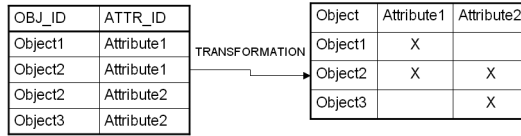


Fig. 8. Transformation of distributional formal context into traditional formal context

This solution allows to connect additional data to distributional formal context. Additional data can be either traditional formal context or distributional formal context. The type of context depends on one reason. Distributional formal context is always parent. Relationships between contexts are generally represented in context rectangles by a line. Context is parent when the line ends with dot.

Logical data model where the relations between two formal contexts are represented is shown in Figure 9. One is distributional formal context and the other is traditional formal context.

Logical data model where the relations between two distributional formal contexts are represented is shown in Figure 10.

Our proposed method is deriving *formal concepts* from *distributional formal context* with the following routine (first method when Context 1 is the main and used for data analysis; Context 2 will be used for saved additional data):

- 1) pick a set of objects A from Context 1;
- 2) derive the attributes A' from Context 1;
- 3) derive all formal concepts B' from Context 2;
- 4) derive (A)' from Context 1;
- 5) derive all formal concepts B'' from Context 2;
- 6) (B'', B') is a *formal concept*.

Let us present an example (Fig. 9) of generating formal concept from distributional formal context (where A is a distributional formal Context 1 and B is an additional formal Context 2):

- 1) pick any set of objects A from Context 1, e.g.  $A = \{\text{Object 3}\}$ ;
- 2) derive the attributes  $A' = \{\text{Attribute 3, Attribute 4, Attribute n}\}$ ;
- 3) derive all attributes B' (has relation with attributes from Context 1) from Context 2, e.g.  $B' = \{\text{Attribute 3} (\{\text{Object 1}\}, \{\text{Attribute 1, Attribute 2}\}), \text{Attribute 4} (\{\text{Object 2}\}, \{\text{Attribute 1, Attribute 3}\}), \text{Attribute n} (\{\text{Object 3, Object 4}\}, \{\text{Attribute 3, Attribute 4}\})\}$ ;
- 4) derive  $(A)' = \{\text{Attribute 3, Attribute 4, Attribute n}\} = \{\text{Object 3, Object 4}\}$ ;
- 5) derive all objects B'' (has relation with objects from Context 1) from Context 2, e.g.  $B'' = \{\text{Object 3} (\{\text{Object 1}\}, \{\text{Attribute 5, Attribute 6}\}), \text{Object 4} (\{\text{Object 5}\}, \{\text{Attribute 1, Attribute 2}\})\}$ ;
- 6)  $(B'', B') = (\{\text{Object 3} (\{\text{Object 1}\}, \{\text{Attribute 5, Attribute 6}\}), \text{Object 4} (\{\text{Object 5}\}, \{\text{Attribute 1, Attribute 2}\})\}, \{\text{Attribute 3} (\{\text{Object 1}\}, \{\text{Attribute 1, Attribute 2}\}), \text{Attribute 4} (\{\text{Object 2}\}, \{\text{Attribute 1, Attribute 3}\}), \text{Attribute n} (\{\text{Object 3, Object 4}\}, \{\text{Attribute 3, Attribute 4}\})\})$  is a *formal concept* from Context 1.

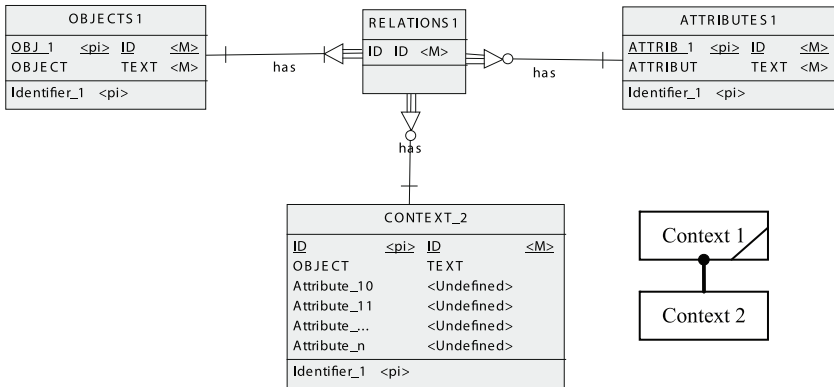


Fig. 9. Relationship between traditional formal context and distributional formal context

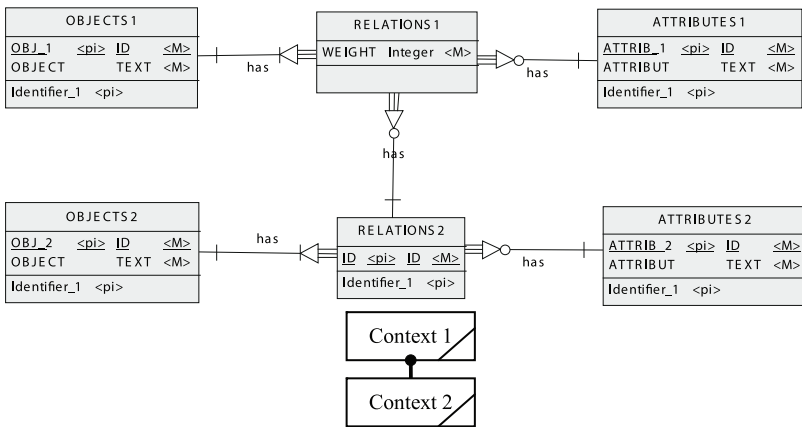


Fig. 10. Relationship between two distributional formal contexts

The second method of using distributional formal context is when parameters to select the specific context from Context 1 are described in Context 2. An example of the method is given in the next section.

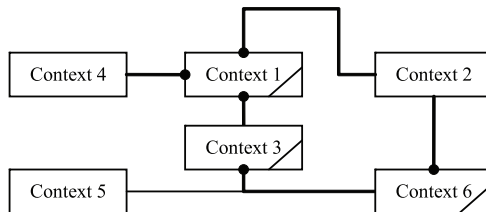


Fig. 11. Network of contexts

The proposed method theoretically and practically allows to compose the network from contexts (Fig. 11). We can see from this network example that some contexts are

traditional (Contexts 2, 4, 5) and others are distributional (Contexts 1, 3, 6). One stage of the future work is to research and analyse the network of contexts.

### Distributional Formal Context for Collecting Many Contexts

The second method of using distributional formal context originates when parameters for selecting the specific context from Context 1 are described in Context 2. The Context 1 is multiple context and it means that from Context 1 we can get many contexts.

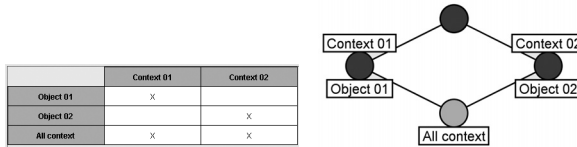


Fig. 12. Contexts described in Context 2

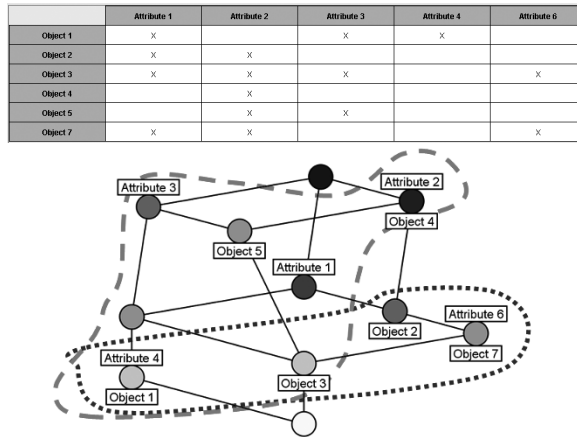


Fig. 13. Context 1 includes Context 01 and Context 02 (dashes show concepts of Context 2; squared dots show concepts of Context 1)

Let us present an example (Fig. 13) of generating formal concept from distributional formal context (where A is a distributional formal Context 1 and B is an additional formal Context 2):

- 1) pick any set of objects A from Context 2, e.g.  $A = \{Object\ 01\}$ ;
- 2) derive the attributes (from Context 2)  $A' = \{Context\ 01\}$ ;
- 3) derive  $(A')' = \{Object\ 01\}' = \{Context\ 01\}$ ;
- 4)  $(A'', A') = (\{Object\ 01\}, \{Context\ 01\})$  is a formal concept of formal Context 2;
- 5) pick any set of objects B from Context 1 (has relation with formal concept  $(A'', A')$  from Context 2 (Table 1)), e.g.  $B = \{Object\ 2\}$ ;



- 6) derive the attributes from Context 1 (has relation with formal concept (A'',A') from Context 2 (Table 1)) B' = {Attribute 1, Attribute 2};
- 7) Derive (B')' = {Object 2 }' = {Attribute 1, Attribute 2};
- 8) (B'',B') = ({Object 2},{Attribute 1, Attribute 2}) is a *formal concept* of formal Context 1.

Figure 14 presents the lattice when we have selected all formal concepts from Context 1 (when Context 02 is selected (generated) from Context 1).

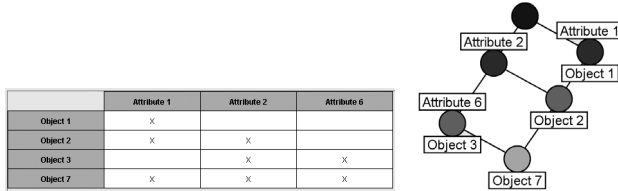


Fig. 14. Context 1 and lattice of Context 1

If we set the Object 2 from Context 2, we obtain lattice shown in Figure 15.

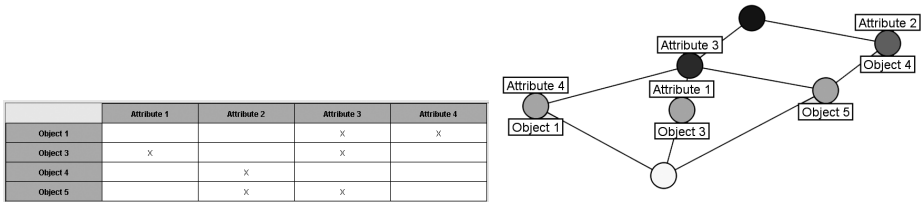


Fig. 15. Context 2 and lattice of Context 2

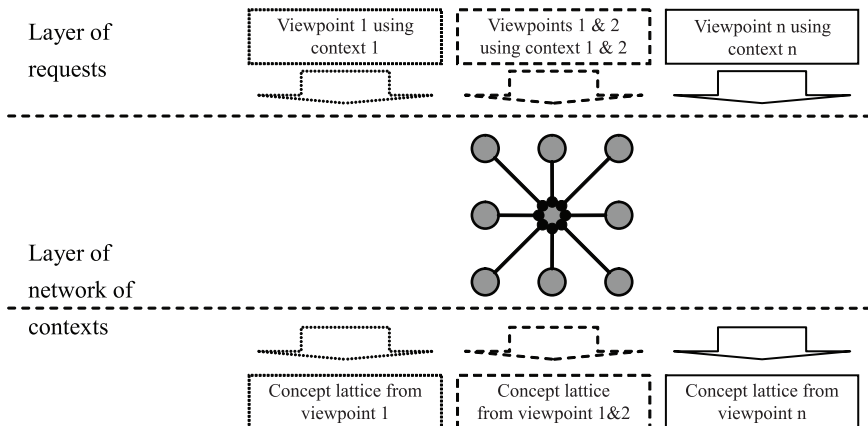


Fig. 16. Diagram of the process where usage of the context network is shown

Table

**Relations between objects and attributes of Context 1 and objects of Context 2**

Objects (Context 1)	Objects (Context 2)	Objects (Context 1)	Objects (Context 2)
Object 1	All context	Attribute 1	All context
Object 2	Context 01	Attribute 2	All context
Object 3	All context	Attribute 3	Context 02
Object 4	Context 02	Attribute 4	Context 02
Object 5	Context 02	Attribute 6	Context 01
Object 7	Context 01		

The process of using distributional formal context for selecting the specific contexts from multiple formal contexts is presented in Figure 16.

### The Experiment

The program agent (Fig. 17) for information search tasks was used for the experiment. Its task was to collect information (from a specific domain) from Internet resources.

While developing the ontology, a set of notices of real estate were explored and analysed.

These notices were acquired from the following websites: <http://www.aruodas.lt>, <http://www.skelbiu.lt>, <http://www.domoplius.lt>, <http://www.alioreklama.lt>, <http://www.skelbimai.lt>, <http://www.edomus.lt>, <http://www.enamai.lt>, <http://www.city24.lt>, <http://www.namai.lt>; <http://www.muge.lt>.

Web Scraper Plus+ tool was used for data acquisition from the websites. The data was collected and saved in a database.

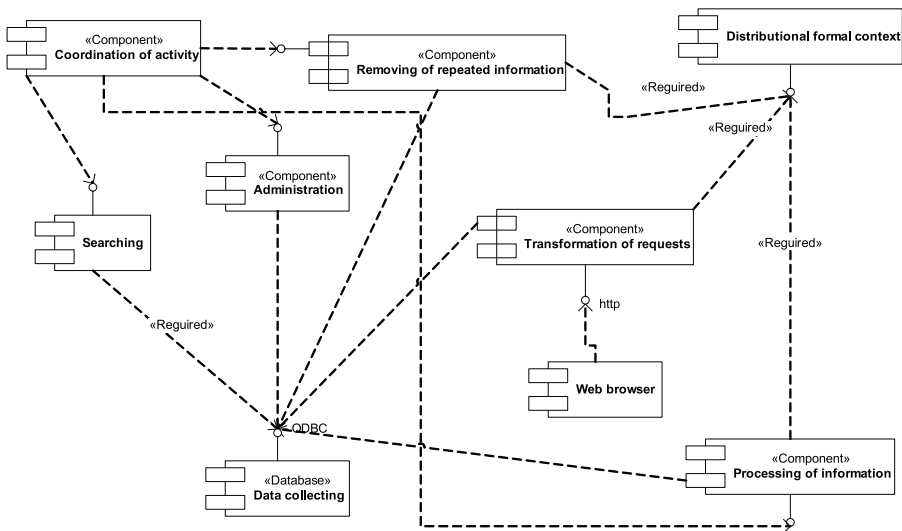
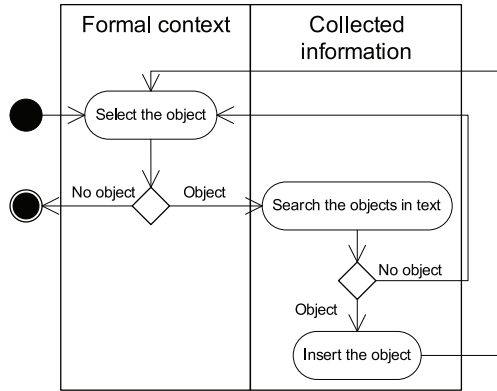


Fig. 17. Architecture of program agent for information search

During the information-processing step, objects (concepts) are acquired from collected information (Fig. 18). To perform this task, formal context and ontology of real estate are needed.

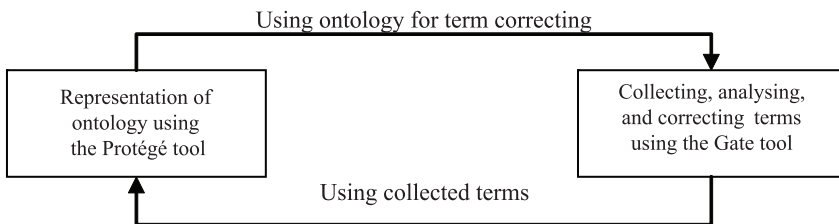


*Fig. 18. Activity diagram of information processing*

Next, we will describe how ontology and formal context was made. Two ontologies were developed: one – using the common method, and the other – using our proposed method (formal context).

**Development of Ontology Using the Common Method**

Common ontology method was used to develop the first ontology (Fig. 1). Protégé 3.2 and Gate 3.1 tools were used for the process of analysing and correcting terms. The terms were collected, then the text was analysed and annotated using the Gate tool (Fig. 19). The developed ontology was presented with the Protégé tool (Fig. 20).



*Fig. 19. The process of development of the ontology of real estate*

**Creating the Classes**

After the analysis of collected concepts, the tree of concepts was created (Fig. 20): real estate, location, action with real estate. Every concept has a lower-level concept.

The class “Real estate” has homes; flats; condition.

The class “Action with real estate” has sell; buy; rent; change.

The class “Location” has

- Vilnius:
  - Naujamiestis;
  - Fabijoniškės;
  - Šnipiškės;
  - Karoliniškės;
- Klaipėda:
  - Laukininkai;
  - Alksninė;
  - Pietinis;
- Kaunas:
  - Šilainai;
  - Dainava;
  - Žaliakalnis etc.

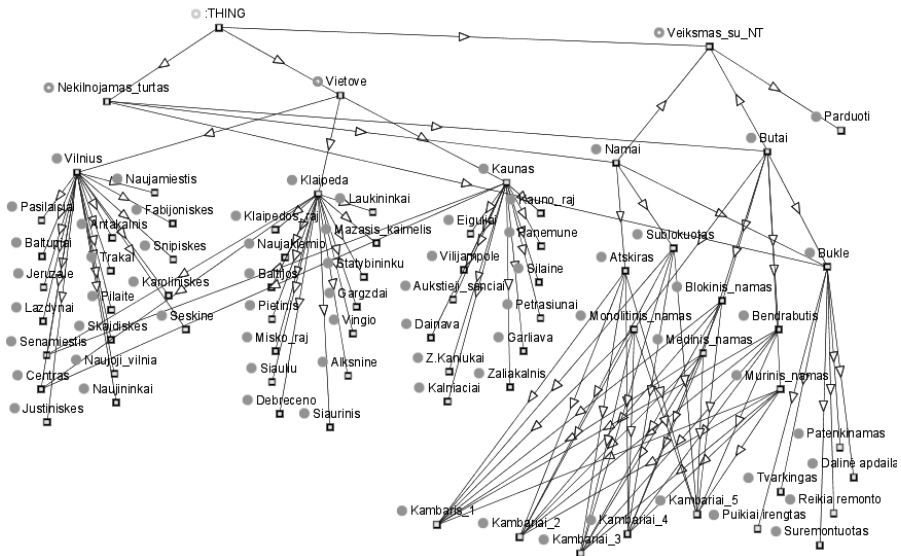


Fig. 20. A hierarchical diagram of the real estate ontology

## Development of Ontology (Formal Context) Using the Proposed Method

The tool “Search of real estate system” was developed to carry out the experiment (Fig. 5, 21). This tool allows to fill the formal context in a database using a web browser. Searching and data analysis forms are created by using formal context. All data for creating these forms is saved in the formal context. That solution allows to change the content of forms dynamically. The tool was developed in PHP language and MySQL database was selected. 11 465 real estate commercials were used to perform the experiment. The main task was “Search”. Here we present more descriptions.

## Formal Context Filling

“Search of real estate system” tool was developed (Fig. 5, 21) for formal context filling. The tool was created in PHP language and MySQL database was selected. This tool can be used to create search criteria and administrate the formal context. It allows to edit the formal context.

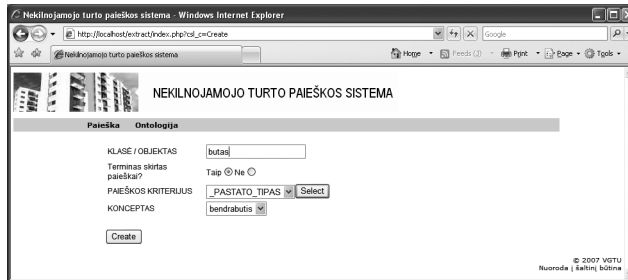


Fig. 21. Real estate system: inserting object in the formal context

ToscanaJ tool was used to analyze the data collected in the formal concept. ToscanaJ is a pure viewer/browser for conceptual schemes; it is optimized for a non-expert audience, it comes with additional tools for creating the data displayed and offers options of additional and more technical analysis. The four main tools are:

- ToscanaJ: the viewer/browser component;
- Elba: an editor for conceptual schemes on relational databases. Database-aware and offering extra tools like exporting SQL scripts;
- Siena: in many ways similar to Elba (mostly thanks to shared code), Siena edits conceptual schemes that store their data in memory;
- Lucca: an experimental editor that is supposed to make use of implication analysis of SQL clauses to allow very explorative and intuitive creation of database-connected systems.

ToscanaJ is the most prominent, however, possibly the most important program of the ToscanaJ suite. It is a very advanced viewer for conceptual schemes that is able to display information queried from the database in lattice diagrams or just using memory-mapped data structures (Fig. 4). There is an opportunity to insert the new concept into formal context quickly (Fig. 17).

The form hyperlink “INSERT” means that there are no searched concepts in the formal context. By clicking the hyperlink “INSERT”, the concept can be inserted into the formal context (Fig. 21). Then information is processed.

## Results and Conclusions

The review of ontology and formal concept structures shows that they are quite similar and both are represented in a hierarchical form. Lightweight ontology has classes, attributes, and relationships. Objects, attributes and relationships are used for formal concept analysis as well. Nevertheless, heavyweight ontology allows to describe

the domain in more detail because it has function terms, restrictions, rules, and axioms. Heavyweight ontology is extensively axiomatized and lightweight ontology often is presented as simple taxonomic structures, and is either slightly or not at all axiomatized.

This paper has proposed a structure of distributional formal context. That solution allows to save more data in the context and to simplify the usage of context in information systems. Two methods of using distributional formal contexts are described in this paper. One of these methods allows to connect the many contexts and to use multi-formal context in the network.

Performing the experiment using the method that realised regular process, concepts were collected and corrected in 1 week. Concepts were collected and corrected in 3 hours when the experiment was performed using our proposed method. Please note that the time of development of “Search of real estate system” tool is not included into evaluation. We also would like to stress that for performing the first experiment, an additional analyst was needed. To perform the second experiment, only the user was needed and the analyst could have been needed if another type of ontology was developed.

The experiment showed that the proposed method allows to search the terms (objects) quickly while analysing a large amount of data. We can show the hierarchical tree (structure) of concepts by using data collected in the formal context. Later, we can also develop ontology from the collected data using another representation language. That allows searching for the dependences, matching, repeated structures, exceptions, etc.

## References

1. Bugaite D., Vasilecas O. Ontology-Based Elicitation of Business Rules. In: A. G. Nilsson et al. (eds.) Proc. of the Fourteenth International Conference on Information Systems Development 2005, *Advances in Information Systems Development: Bridging the Gap between Academia and Industry*. Karlstad, Sweden, 15–17 August, 2005, Springer, 2006, p. 795–806.
2. Dau F. (2004) Types and Tokens for Logic with Diagrams. In: K. E. Wolff, H. Pfeiffer, & H. Delugach (eds.), *Conceptual Structures at Work: 12th International Conference on Conceptual Structures*. Berlin: Springer, p. 62–93.
3. Ganter B., Wille R. (1998) Applied Lattice Theory: Formal Concept Analysis. In: G. Grätzer: *General lattice theory*. 2<sup>nd</sup> ed. Birkhäuser Verlag, Basel.
4. Ganter B., Wille R. (1999) *Formal Concept Analysis: Mathematical Foundations*. Berlin-Heidelberg: Springer.
5. Godin R., Gecsei J., Pichet C. (1989) Design of browsing interface for information retrieval. In: N. J. Belkin & C. J. van Rijsbergen (eds.), *Proc. SIGIR '89*, p. 32–39.
6. Guarino N. (1997) Understanding, building and using ontologies. *Int. Journal of Human-Computer Studies*, vol. 45, no. 2/3 (Feb/Mar).
7. Lavbic D., Krisper M. Rapid Ontology Development Model Based On Business Rules Management Approach For The Use In Business Applications. Proceedings of the 6<sup>th</sup> ICEIS Doctoral Consortium (DCEIS 2008) in conjunction with 10<sup>th</sup> International Conference on Enterprise Information Systems (ICEIS 2008), Cardoso, Jorge (ed.), Barcelona, Spain, pp. 24–35 (81).
8. Ling Liu and M. Tamer Özsu (eds.) (2008) *Encyclopedia of Database Systems*, Springer-Verlag.
9. Margolis E., Laurence S. (2007) The Ontology of Concepts – Abstract Objects or Mental Representations? *Nous*, vol. 41, issue 4, p. 561.
10. Martin J., Odell J. (1994) *Object-Oriented Methods: A Foundation*. Prentice-Hall, p. 52.
11. Mayers A., Maulsby D. (2004) Glossary (Online). Available: <http://acypher.com/wwid/BackMatter/Glossary.html>.
12. Sasa A., Matjaz B. J., Krisper M. Service-oriented framework for human task support and automation. *IEEE transactions on industrial informatics*. Nov. 2008, vol. 4, no. 4, p. 292–302.

13. Tilley T. (2004) *Formal Concept Analysis Application to Requirements Engineering and Design*. School of Information Technology and Electrical Engineering, University of Queensland, PhD Thesis, December. Available: <http://www.tomtilley.net/publications/tilley04formal.pdf>
14. Vasilecas O., Bugaite D. Ontology-based Information Systems Development: the Problem of Automation of Information Processing Rules. In: E. Neuhold, T. Yakhno (eds.), Proc. of the Fourth International Conference Advances in Information Systems (ADVIS'2006), Izmir, Turkey, 18–20 October, 2006, Springer, LNCS 4243, p. 187–196.
15. Wikipedia (2008) Concept. Wikipedia: The Free Encyclopedia. [Accessed on 2008-02-22]. <http://en.wikipedia.org/wiki/Concept>.
16. Wikipedia (2008) Ontology (computer\_science). Wikipedia. The Free Encyclopedia. Available: [http://en.wikipedia.org/wiki/Ontology\\_\(computer\\_science\)](http://en.wikipedia.org/wiki/Ontology_(computer_science))
17. Wille R. (1982) *Restructuring Lattice Theory: an approach based on hierarchies of concept*. Ordered sets (editor I. Rival). Reidel, Dordrecht-Boston, p. 445–470.
18. Wille R. (1992) Concept Lattices and Conceptual Knowledge Systems. *Computers & Mathematics with Applications*, 23, 493–515.
19. Wille R. (1997a) Introduction to Formal Concept Analysis. In: G. Negrini (ed.), *Modelli e modellizzazione*. (Models and modelling.) Consiglio Nazionale delle Ricerche, Istituto di Studi sulla Ricerca e Documentazione Scientifica, Roma, 39–51.
20. Wille, R. (1997b) Conceptual Graphs and Formal Concept Analysis In: Dickson Lukose, Harry Delugach, Mary Keeler, Leroy Searle, and John F. Sowa (eds.), *Conceptual Structures: Fulfilling Peirce's Dream*, Proc. of the Fifth Int. Conf. on Conceptual Structures (ICCS'97), August 3–8, University of Washington, Seattle, USA, LNAI, Number 1257, Springer Verlag, Berlin, 290–303.
21. Wille R. (2001) Why Can Concept Lattices Support Knowledge Discovery in Databases? *ICCS'01 International Workshop on Concept Lattices-Based KDD*, 7–20.
22. Wolf K. (1993) A first course in formal concept analysis. In: Faulbaum, F. (ed.), *SoftStat '93 Advances in Statistical Software 4*, p. 429–438.
23. WORDNET (2008) A lexical database for the English language [Online]. Available: <http://wordnet.princeton.edu/perl/webwn?s=concept>

## Automatic Verification of the Conceptual Model and Its Documentation

Algirdas Laukaitis<sup>1</sup> and Olegas Vasilecas<sup>1,2</sup>

<sup>1</sup> Vilnius Gediminas Technical University, Sauletekio al. 11, LT-10223 Vilnius-40, Lithuania  
*algirdas.laukaitis@fm.vgtu.lt, olegas@fm.vgtu.lt*

<sup>2</sup> Klaipeda University, Herkaus Manto 84, LT-92294 Klaipeda, *olegas.vasilecas@ik.ku.lt*

By using background knowledge of general and specific domains, and by processing a new natural language corpus, experts are able to produce a conceptual model for some specific domain. In this paper, we present a model that tries to capture some aspects of this conceptual modeling process. This model is functionally organized into two information processing streams: one reflects the process of formal concept lattice generation from the domain conceptual model; the other reflects the process of formal concept lattice generation from the domain documentation. It is expected that similarity between those concept lattices reflects similarity between documentation and the conceptual model. In addition to this process of formal verification, a set of natural language processing artifacts are created. Those artifacts then can be used for the development of information system natural language interfaces. To demonstrate it, an experiment of concept identification from natural language queries is provided at the end of this paper.

**Keywords:** information systems engineering, formal concept analysis, IS document self-organization, natural language processing.

### 1 Introduction

Software engineers spend hours in defining information systems (IS) requirements and finding common ground of understanding. The overwhelming majority of IS requirements are written in a natural language supplemented with a conceptual model and other semi-formal UML diagrams. In the form of semantic indexes, the bridge between documents and the conceptual model can be useful for more effective communication and model management. Therefore, integration of the natural language processing (NLP) into information system documentation process is an important factor for meeting the challenges of modern software engineering methods. Reusing natural language IS requirement specifications and compiling them into formal statements has been a prolonged challenge [2], [17]. Kevin Ryan claimed that NLP is not mature enough to be used in requirement engineering [16] and our research justifies that as well. Nevertheless, we hope that the current paper will suggest some promising findings towards this challenging task.



In this paper, by combining the symbolic and connectionist paradigms, we present our efforts to overcome difficulties and problems of the natural language usage in all stages of IS development. The self-organizing map (SOM) [13] is proposed as a tool to analyze the documents and communication utterance, and Formal Concept Analysis (FCA) [5] is suggested as a method to reinterpret SOM topology and to verify the comprehensibility and soundness of the information system documentation and model. All presented ideas and methodological inference have been tested with the IBM Information FrameWork (IFW) [9], which is a comprehensive set of banking-specific business models from the IBM corporation. For our research, we have chosen the set of models under the name *Banking Data Warehouse*. We define the following problems: 1) *How can we formally verify the IS documentation if we have at least several sentence description for each business information system component?* 2) *What is the architectural solution of the system where the designers, modelers, requirement engineers can verify new pieces of textual documentation and automatically generate hierarchical prototypes of the information system model?* 3) *What components from the new modeling system can be taken and reused as plugins in the natural language interfaces (i.e. database querying [1])? It must be proved on an experimental basis that those components can compete with the existing natural language systems.*

The solutions to the stated problems organize the rest of the paper as follows: first, we present the general framework of an automated model generation system from the IS documentation and utterances by engineers. Next, we present the IBM's IFW solution and the model which we used in our experiments. We present FCA as the formal technique to analyze the IS model on the *object:attribute* sets. In Section 4, we present the architectural solution of the natural language processing (NLP) system which was built from open source, state-of-the-art NLP components. Then we present an idea of the conceptual model vector space. The motivation for introducing this stage to the modeling process is that it helps us deal with the modeling documentation and its topological structures numerically. Then the SOM of the conceptual model is introduced in Chapter 5. Finally, to prove the soundness of the proposed method, we provide a numerical experiment in which the ability of the system to identify concepts from user utterance is tested. The IBM Voice Toolkit for WebSphere [10] (an approach based on statistical machine learning) solution is compared with the system suggested in this paper.

## 2 The General Framework of the Solution

Conceptual models offer an abstract view on certain characteristics of the domain under consideration. They are used for different purposes – such as a communication instrument between users and developers, for managing and understanding the complexity within the application domain, etc. The presence of tools and methodology that supports integration of the requirement documents and communication utterance into conceptual model development is crucial for a successful development of the IS architectural framework.

In this paper, we suggest the use of SOM to classify IS documentation and IS utterance on a supervised and unsupervised basis. SOM has been extensively studied in the field of textual analysis. Such projects as WEBSOM [11] [14] have shown that the SOM algorithm can organize very large text collections and that SOM is suitable for

visualization and intuitive exploration of the document collection. The experiments with the Reuters corpus (a popular benchmark for text classification) were investigated in [8]; there was evidence that SOM can outperform other alternatives.

Nevertheless, in the field of IS modeling the connectionist paradigm has been met with some skepticism. The reason is that IS architects and modelers want to give the credibility on how clusters received from document processing are related and explain the semantic meaning of the underlying topology of documents. To overcome this problem, we suggest that FCA can give more on that account by formally analyzing the set of objects and their attributes. On the other hand, when directly applied to the large data set of textual information, FCA is of little meaning for the presentation of the overwhelming lattice. Those arguments motivate integration of FCA and other text clustering techniques. In that sense, our work bears some resemblance with the work of Hotho et al. [7]. They used BiSec-kk-Means algorithm for text clustering and then FCA was applied to explain relationships between clusters. Authors of the paper have shown the usability of such approach in explaining the relationships between clusters of the Reuters-21578 text collection.

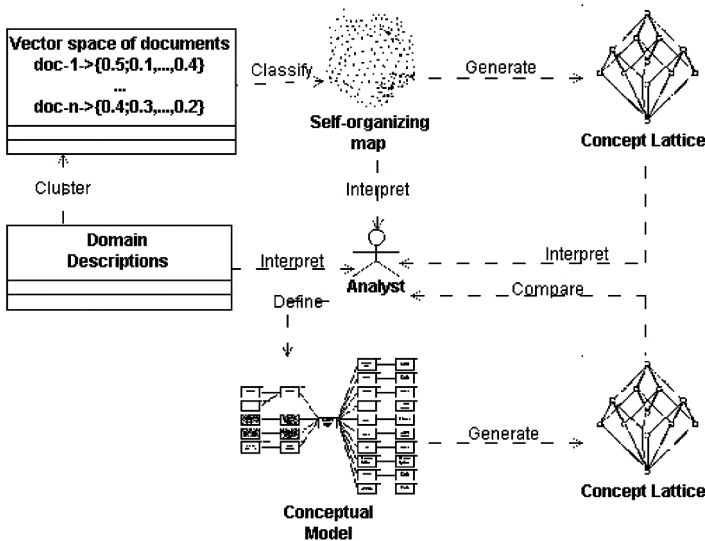


Fig. 1. Process of integration: conceptual modeling, textual description cluster detection and interpretation by use of FCA

Our approach differs in two important respects. First, our goal is not text clustering. Our goal is automated generation of the ontology from textual documents if there is no knowledge base produced by human experts. In case the knowledge base has already been developed, we seek for the method that formally measures the comprehensibility of the knowledge base topology and in case of new documents and concepts, automatically integrates them into the knowledge base.

The overall process of automatically clustering concept descriptions and then deriving concept hierarchies from SOM is presented in Figure 1. First, the corpus is created from the model concept descriptions; it is called domain descriptions in

Figure 1. Then vector space of the corpus is created using natural language processing framework, domain ontology, and WordNet ontology [15]. SOM is built and used for cluster analysis. Next, with conceptual context and concept lattice (CL) improvements are made in the understanding of cluster relationships. Simultaneously, CL is created directly from the conceptual model. The analyst can compare the lattice received from IS documentation and the lattice generated from the conceptual model. If both lattices are similar, we can say that the quality of IS documentation is acceptable.

### 3 Business Knowledge Bases and Formal Concept Analysis

The problem with data-centric enterprise-wide models is that they are difficult to understand. Their abstract and generic concepts are unfamiliar to both business people and IS professionals, and remote from their local organizational contexts [4]. Natural language processing and understanding techniques can be used to solve the mentioned problems. Nevertheless, before applying the NLP techniques in IS engineering, we must have some formal method to deal with the sets of  $\{classes, object, and attributes\}$  which are the products of systems of natural language processing. In this section we introduce FCA as the method for automatical building of a hierarchical structure of concepts (or classes) from the  $\{object:attribute\}$  set.

In Figure 2 (left side) we can see an excerpt from the IBM IFW financial services data model (FSDM) [9], which is a domain-specific model based on the ideas of the experts of the IBM financial service solutions center. The IBM financial services data model consists of a high level strategic classification of domain classes integrated with particular business solutions (e.g., Credit Risk Analysis) and logical and physical data entity-relationship (ER) models.

CL of the extract from the model have been produced by FCA with Galicia software [19] and are shown on the right side of Figure 2. As we can see, it is consistent with the original model. It replicates the underlying structure of the conceptual model originally produced by a human expert team; in addition, it suggests one formal concept that aggregates *Arrangement* and *Resource Item*: the two top concepts of the original model.

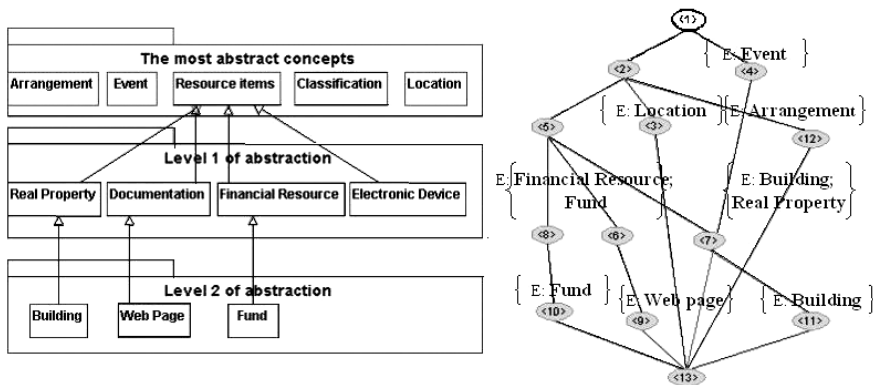


Fig. 2. Left side: a small excerpt from the financial services conceptual model. Right side: CL from this conceptual model. (We see that FCA depicts the structure of the conceptual model.)

FCA is used to represent underlying data in the hierarchical form of the concepts. The most adapted form in the FCA analysis for data representation is CL. Due to its comprehensive form in visualising the underlying hierarchical structure of the data and rigorous mathematical formalism, FCA has developed into a fully-fledged theory for data analysis since its introduction in the 1980s [5]. FCA has been successfully applied in many areas, but our interest in this paper is the ability to use it in the area of IS modeling. In defining the concepts and attributes, FCA is similar to the database theory and object orientated system design. Due to this fact, FCA has been often applied in class diagram design in IS [5].

For the introduction to the area of FCA, we can return to Figure 2. The conceptual model extract in the figure has 12 objects. Let us name them the set  $G$ . Let  $M$  be the set of attributes that characterise the set of objects, i.e., an attribute is included into the set  $M$  if it is an attribute for at least one object from the set  $G$ . In our example we have 137 attributes (the whole model has more than 1000 objects and more than 4000 attributes). We identify the index  $I$  as a binary relationship between two sets  $G$  and  $M$ , i.e.,  $I \subseteq G \times M$ . In our example the index  $I$  will mark that, eg., an attribute “Interest Rate” belongs to an object “Arrangement” and that it does not belong to an object “Event”.

In order to be able to start FCA algorithms, we define a triple  $\mathbb{K} := (G, M, I)$  which is called a formal context. Further, we define subsets  $A \subseteq G$  and  $B \subseteq M$  as follows:

$$A' := \{m \in M | (g, m) \in I \text{ for all } g \in G\};$$

$$B' := \{g \in G | (g, m) \in I \text{ for all } m \in B\}.$$

Then a formal concept of a formal context  $(G, M, I)$  is defined as a pair  $(A, B)$  with  $A \subseteq G$ ,  $B \subseteq M$ ,  $A' = B$  and  $B' = A$ . The sets  $A$  and  $B$  are called extend and intend of the formal concept  $(A, B)$ . The set of all formal concepts  $\mathfrak{B}(\mathbb{K})$  of a context  $(G, M, I)$  together with the partial order  $(A_1, B_1) \leq (A_2, B_2) : \Leftrightarrow A_1 \subseteq A_2$  is called the concept lattice of the context  $(G, M, I)$ .

In Figure 2, the FCA algorithm *Incremental Lattice Builder* generated 11 formal concepts. In the lattice diagram, the name of an object  $g$  is attached to the circle and represents the smallest concept with  $g$  in its extent. The name of an attribute  $m$  is always attached to the circle representing the largest concept with  $m$  in its intent. In the lattice diagram an object  $g$  has an attribute  $m$  if and only if there is an ascending path from the circle labeled by  $g$  to the circle labeled by  $m$ . The extent of the formal concept includes all objects whose labels are below in the hierarchy, and the intent includes all attributes attached to the concepts above. For example, the concept 7 has  $\{Building; Real Property\}$  as extend (the label  $E$ : in the diagram), and  $\{Postal Address; Environmental Problem Type; Owner; \dots etc\}$  as intent (due to the huge number of attributes they are not shown in the figure).

## 4 Vector Space Representation of the Conceptual Model

The vector space model (VSM) for document transformation into vectors is a well-known representation approach that transforms a document into a weight vector in automatic text clustering and classification. The method is based on the bag-of-words approach, which ignores the order of words in a sentence and uses basic occurrence information [18].

On the other hand, the vector space model’s dimensionality is based on the total number of words in the data set and it brings difficulties for the large data sets. The document corpus of the conceptual model described above included 3587 words. The process of dimensionality reduction and noise filtering is depicted in Figure 4. All presented processes are described in detail below.

1) *Transform conceptual model*. As the first step we transform conceptual model to the Web Ontology Language (OWL) structure. The motivation behind this step is that the OWL is one of the most used standard in describing the knowledge base and we already use it in Semantic Web applications. Additional motivation for using OWL is the availability of the knowledge base development tools such as Protégé – OWL editor [12] that supports OWL standard.

2) *Extract triplet*. The triplet consists of concept name, the most abstract parent concept name – class label for a particular document, and description of the concept. To be more specific, the following steps were performed: first, we selected only concepts (entities) from ‘C’ level of the conceptual model and then selected the textual description of each entity. We received 1256 documents in the corpus, each document describing one concept. Each document in the corpus has been labeled with its original concept name and its top parent concept name. For example, the concept “Employee” has the following entry in the corpus: { *Concept-Employee; Parent-Individual; Top parent concept – Involved Party ; Description – An Employee is an Individual who is currently, potentially or previously employed by an Organization, commonly the Financial Institution itself...*  }. We had to add textual descriptions to 254 concepts. It was done because we wanted to measure additional documentation impact on the classification accuracy of concepts. The descriptions were taken from web dictionaries. 198 concepts were removed due to short textual descriptions and our inability to supplement them from the web dictionaries. After these steps, we obtain our final corpus for the evaluation. It consists of the 1058 documents, distributed over 9 top parent concepts (*involved party, products, arrangement, event, location, resource items, condition, classification, business*).

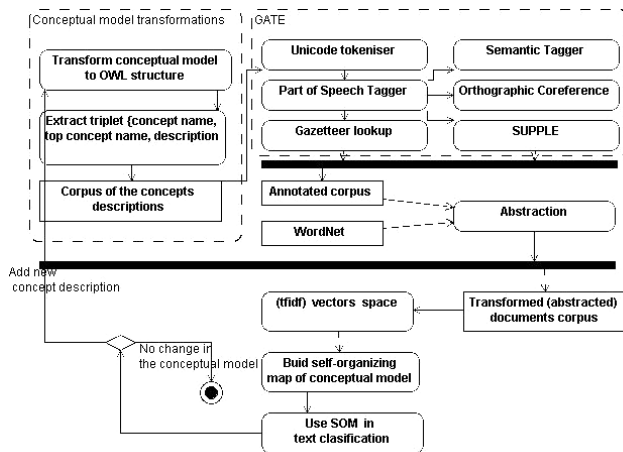


Fig. 3. The processes of dimensionality reduction and the conceptual model SOM design

3) GATE – Natural Language Processing Engine is a well-established infrastructure for customization and development of NLP components [3]. It is a robust and scalable infrastructure for NLP and it allows users to use various modules of NLP as plugins. We briefly describe modules used in our research for building vector spaces of concepts. The Unicode tokeniser splits the text into simple tokens. The tagger produces a part-of-speech tag as an annotation for each word or symbol. The gazetteer further reduces dimensionality of the document corpus prior to classification. The semantic tagger provides finite state transduction over annotations based on regular expressions. It produced an additional set of named entities, and we replaced each named entity with the class label. Orthographic Coreference module adds identity relations between named entities found by the semantic tagger. SUPPLE is a bottom-up parser that constructs syntax trees and logical forms of English sentences. We used it only to remove tokens not annotated by this module. All modules within GATE produced annotations – pairs of nodes pointing to positions inside the document content, and a set of attributes-values, encoding linguistic information.

4) *Abstraction*. The basic idea of the abstraction process is to replace terms by more abstract concepts as defined in a given thesaurus in order to capture similarities at various levels of generalization. For this purpose we used WordNet [15] and annotated GATE corpus as the background knowledge base. WordNet consists of the so-called synsets, together with a hypernym/hyponym hierarchy [6]. To modify the word vector representations, all nouns have been replaced by the corresponding concept of WordNet ('synset'). Some words have several semantic classes ('synsets') and in that case we used a disambiguation method provided by WordNet – the most common meaning for a word in English was our choice. The words replaced by the GATE named entities annotation scheme were not included in the WordNet processing.

5) *Vector space*. In our experiments we used vector space of the terms vectors weighted by *tfidf* (term frequency inverse document frequency) [18], which is defined as follows:

$$tfidf(c,t) = tf(c,t) \times \log \frac{|C|}{|C_t|}.$$

where  $tf(c,t)$  is the frequency of term  $t$  in concept description  $c$ , and  $C$  is the total number of terms, and  $C_t$  is the number of concept descriptions containing this term.  $tfidf(c,t)$  weighs the frequency of a term in a concept description with a factor that discounts its importance when it appears in almost all concept descriptions.

## 5 Self-Organizing Map of the IS Conceptual Model

Neurally inspired systems, also known as the connectionist approach, replace the use of symbols in problem solving by using simple arithmetic units through the process of adaptation. The winner-take-all algorithms, also known as the self-organizing network, select the single node in a layer of nodes that responds most strongly to the input pattern. In the past decade, SOM have been extensively studied in the area of text clustering. The ideas and results presented here are of general purpose and could be applied in knowledge development by means of the connectionist paradigm in general.

SOM consists of a regular grid of map units. Each output unit  $i$  is represented by A prototype vector,  $m_i = [m_{i1}...m_{id}]$ , where  $d$  is input vector dimension. Input units take the input in terms of a feature vector and propagate the input onto the output units. The number of neurons and topological structure of the grid determines the accuracy and generalization capabilities of SOM.

During learning the unit with the highest activation, i.e. the best matching unit regarding a randomly selected input vector is adapted in a way that it will exhibit even higher activation regarding this input in the future. Additionally, the units in the neighborhood of the best matching unit are also adapted to exhibit higher activation regarding the given input.

As a result of training SOM with the text corpora of the IBM IFW financial warehouse conceptual model, we obtain a map which is shown in Figure 4. SOM has been trained for 100,000 learning iterations with learning rate initially set to 0.5. The learning rate decreased gradually to 0 during the learning iterations.

Table 1

**Classification accuracy (CA) and average quantization error (AQE) of the conceptual model SOM**

	No hyponym	WordNet synset replacements	One level up hyponym replacements	Two levels up hyponym replacements	Three levels up hyponym replacements
CA	29.57	29.56	41.53	39.27	26.44
ACQ	4.83	4.81	4.56	4.83	4.28

It was expected that if the conceptual model vector space has some clusters that resemble the conceptual model itself, the model will be easier to understand compared with the model of a more random structure. On a closer look at the map, we can find regions containing semantically related concepts. For example, the upper right side of the final map represents a cluster of concepts “Arrangement” and the lower right side “Resource items”. Such map can be used as an interface to the underlying conceptual model. To obtain information from the collection of documents, the users may formulate queries describing their information needs in terms of the features of the required concept.

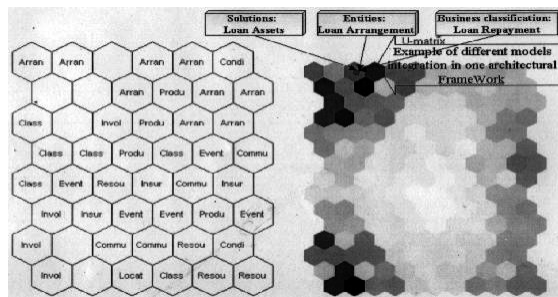


Fig. 4. SOM for the conceptual model. Labels: *invol*, *accou*, *locat*, *arran*, *event*, *produ*, *resou*, *condi* represent concepts: involved party, accounting, location, event, product, resource, condition.

Figure 5 shows the concept lattice computed from SOM shown in Figure 4. We obtain a list of 23 formal concepts. Each of them groups several neurons from SOM. We can find the grouping similarity of the neurons that are located in the neighborhood of each other. On the other hand, some concepts group neurons that are at some distance from each other. The basic idea of this step is that we receive a closed loop in the business knowledge engineering by an artificial intelligent agent. The agent classifies all IS textual information with the help of the SOM technique and then, using FCA, it builds hierarchical knowledge bases. For the details on how to apply FCA in cluster analysis (SOM in our case), we refer to the paper [7]. The paper describes an algorithm which has been used in our research.

The impact of the abstraction and natural language processing on the performance of the information system model can be checked with classification accuracy (CA) measure. It simply counts the minority of concepts at any grid point and presents the count as a classification error. For example, after the training each map unit has a label assigned by the highest number of concepts (Figure 4). In Figure 4, the neuron on the upper left side mapped 4 concepts with the label *Arrangement* and 2 with label *Event*. Thus, classification accuracy for this neuron is 66%. Another metric to measure classification accuracy is average quantization error AQE. It is defined as the average distance between every input vector and its best matching unit:

$$AQE = \frac{1}{N} \sum_{i=1}^N |x_i - b_i|$$

where  $N$  is the total number of input patterns,  $x_i$  is the vector of each pattern and  $b_i$  is best matching unit (BMU) for each pattern  $x_i$ . Findings of the influence of terms abstraction and natural language processing are shown in Table 1.

We can see that the hypernym level one is optimal compared with more abstract concepts. The phenomena can be explained by the fact that different meanings of the term, if too abstract, will be treated as the same and, as a result, semantics of discretionary power will be lost.

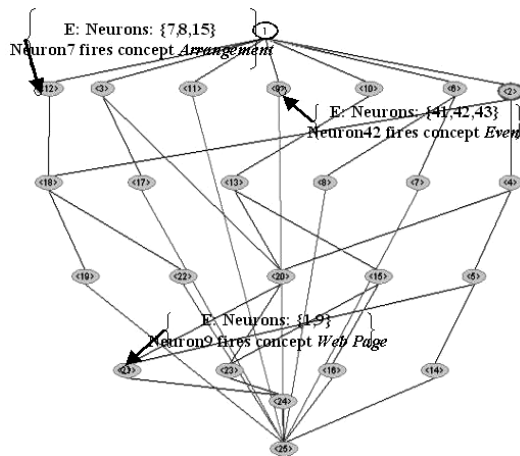


Fig. 5. Concept lattice received from the SOM presented in Figure 4.



## 6 The Experiment

In the previous sections, we have shown how to build a hierarchical conceptual model from IS documentation and how to verify formally the business information system model. In addition, as mentioned in the introduction, one of the objectives of this research project was to find the techniques and tools of IS modeling that give an opportunity to reuse IS model components as the final products in IS natural language interfaces. We argue that such component can be SOM of the IS conceptual model.

Reusing SOM in IS interfaces is quite simple. Each time the sentence is presented to the system, we have one activated neuron which is associated with one concept from the conceptual model. Additionally, we have the set of formal concepts associated with the activated neuron. Both the label of the activated neuron and the set of formal concepts can be used by formal language generation engines (i.e., structured query language (SQL) sentence generator for querying databases). Then the following hypothesis is formulated in this section: *SOM received from IS documentation can compete with the state-of-the-art concept identification solutions currently available in the market.*

The following experiment was conducted to test this hypothesis. IBM WebSphere Voice Server NLU toolbox, which is part of the IBM WebSphere software platform, was chosen as the solution competitive to the one suggested in this paper. From IBM presentation [10] it appeared that the system is primarily intended to support database interfaces in the telecommunication market. It was a challenging task to test it on a more complex system, i.e., a full enterprise conceptual model for the financial market.

SOM of the conceptual model and CL was used as an alternative to the IBM WebSphere Voice Server NLU solution. We adopted the black box approach for both solutions: put the training data, compile, and test the system response for the new data set. The data set of 1058 pairs *textual description:concept name* mentioned above was constructed to train the IBM NLU model. The same set was used to get SOM of the business model.

Then a group consisting of 9 students was instructed about the database model. They had the task to present for the system 20 questions about information related to the concept “Involved Party”. For example, one of the questions was “*How many customers we have in our system?*” We scored the answers from the system as correct if it identified the correct concept “Involved Party”.

Table 2

**Concept identification comparison between IBM NLU toolbox and SOM of the database conceptual model**

	CN = 9	CN = 50	CN = 200	CN = 400	CN = 500
IBM NLU	36.82	17.26	14.82	11.15	8.22
SOM	46.73	30.70	27.11	20.53	18.83
No additional descriptions	38.24	18.43	15.72	12.77	9.52

In the beginning only 9 top concepts were considered, i.e. all 1058 documents were labeled with the most abstract concept names from the conceptual model. For example, documents that described concepts “Loan” and “Deposit” were labeled with

the concept name “Arrangement” because concepts “Loan” and “Deposit” are subtypes of the concept “Arrangement”.

Next, we increased the number of concept names that we put into the model up to 50. For example, documents that described concepts “Loan” and “Deposit” were labeled with “Loan” and “Deposit” names. Then we increased the number of concept names up to 200, 400 and, finally, 500. Table 2 shows the results of the experiment. Columns show the number of concepts. The row named *IBM NLU* represents results for the IBM WebSphere Voice Server NLU toolbox. The row named *SOM* represents results for the SOM of the conceptual model that has been constructed with the method described in this paper. The row named *No additional descriptions* represents results for SOM of the conceptual model without the 254 additional documents mentioned above. To detect the classification error, the proportion of correctly identified concepts was determined.

As we can see, the performance of the IBM system was similar to the SOM response. The behavior of the IBM system is difficult to explain because it is close system and there was no description of algorithms used. For all cases i.e. IBM, SOM and SOM without additional descriptions the performance decreased when the number of concepts increased. The solution that can increase accuracy of concepts identification is suggested by comparing results in the third and second row of Table 2. We see that the 254 descriptions we added to the system significantly improved the response of the system.

## 7 Conclusion

Conceptual models and other forms of knowledge bases can be viewed as products that have emerged from human natural language processing. Self-organization is the key property of human mental activity, and the present research focuses on what self-organization properties can be found in the knowledge base documentation. It has been suggested to build a conceptual model vector space and its SOM by comparing the concept lattice received from a manually constructed conceptual model and the concept lattice received from SOM of the conceptual model. We argued that if both concept lattices resemble each other, we can assert that IS documentation quality is acceptable.

The presented architectural solution for the software developers can be labor-intensive. The payoff of such approach is an ability to generate formal language statements directly from IS documentation and IS user utterance. We have shown that with SOM and FCA we can indicate inadequate concept descriptions and improve the process of knowledge base development. The presented methodology can serve as the tool for maintaining and improving enterprise-wide knowledge bases.

There have been many research projects concerning questions of semantic parsing, i.e., the automatic generation of the formal language from the natural language. But those projects have been concerned only with semantic parsing as a separate stage not integrated into the process of software development. The solution presented in this paper allows us to integrate IS design and analysis stages with the stage of semantic parsing. In this paper, we demonstrated that we can label documents and user questions with concept names of the conceptual model. In the future we hope to extend those results by generating SQL sentences and then querying databases. The present research has shown that if we want to build a comprehensible model, we must give more attention to describing concepts by the natural language.

## References

1. Androutsopoulos I., Ritchie G. D., Thanisch P. Time, Tense and Aspect in Natural Language Database Interfaces. *Natural Language Engineering*, 4, 1998, 229–276.
2. Burg J. F. M., Riet R. P. Enhancing CASE Environments by Using Linguistics. *International Journal of Software Engineering and Knowledge Engineering* 8(4), 1998, 435–448.
3. Cunningham H. GATE, a General Architecture for Text Engineering. *Computers and the Humanities*, 36, 2002, 223–254.
4. Darke P., Shanks G. Understanding Corporate Data Models. *Information and Management* 35, 1999, 19–30.
5. Ganter B., Wille R. *Formal Concept Analysis: Mathematical Foundations*. Springer, Berlin-Heidelberg, 1999.
6. Hofmann T. Probabilistic latent semantic indexing. In: *Research and Development in Information Retrieval*, 1999, 50–57.
7. Hotho A., Staab S., Stumme G. Explaining text clustering results using semantic structures. In: *Principles of Data Mining and Knowledge Discovery, 7th European Conference, PKDD 2003*, Croatia. LNCS. Springer 2003, 22–26.
8. Hung C., Wermter S., Smith P. *Hybrid Neural Document Clustering Using Guided Self-organisation and WordNet*. Issue of IEEE Intelligent Systems, 2004, 68–77.
9. IBM. IBM Banking Data Warehouse General Information Manual. Available from on the IBM corporate site <http://www.ibm.com> (accessed on July 2007).
10. IBM Voice Toolkit V5.1 for WebSphere Studio. <http://www-306.ibm.com/software/> (accessed on July 2007).
11. Kaski S., Honkela T., Lagus K., Kohonen T. WEBSOM self-organizing maps of document collections. *Neurocomputing*, 21, 1998, 101–117.
12. Knublauch H., Ferguson R., Noy N. F. The Protege-OWL plugin: an open development environment for semantic web applications. Third International Semantic Web Conference. ISWC2004, Lecture Notes in Computer Science, 3298. Springer-Verlag: Heidelberg 2004, 229–243.
13. Kohonen T. *Self-Organizing Maps*, Springer-Verlag, 2001.
14. Lagus K., Honkela T., Kaski S., Kohonen T. WEBSOM for textual data mining. *Artificial Intelligence Review*, 13 (5/6) 1999, 345–364.
15. Miller G. A. WordNet: A Dictionary Browser, Proc. of 1st Int'l Conf. "Information in Data", 1985, 25–28.
16. Ryan K. The role of natural language in requirements engineering. *Proceedings of IEEE International Symposium on Requirements Engineering*, IEEE Computer Society Press, 1993, 240–242.
17. Rolland C., Proix C. A Natural Language Approach to Requirements Engineering. 4th International CAiSE Conference, Manchester UK, 1992, 257–277.
18. Salton G. *Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer*. Addison-Wesley, 1989.
19. Valtchev P., Grosser D., Roume C., Rouane H. M. GALICIA: an open platform for lattices. In: A. de Moor, B. Ganter, editors, *Using Conceptual Structures: Contributions to 11th Intl. Conference on Conceptual Structures 2003*, 241–254.

## Ontology Transformation: from Requirements to Conceptual Model\*

Justas Trinkunas<sup>1</sup> and Olegas Vasilecas<sup>2,3</sup>

<sup>1</sup> Information Systems Research Laboratory, Faculty of Fundamental Sciences, Vilnius Gediminas Technical University, Sauletekio al. 11, LT-10223 Vilnius-40, Lithuania, *justas@isl.vgtu.lt*

<sup>2</sup> Information Systems Research Laboratory, Faculty of Fundamental Sciences, Vilnius Gediminas Technical University, Sauletekio al. 11, LT-10223 Vilnius-40, Lithuania, *olegas@isl.vgtu.lt*

<sup>3</sup> Department of Computer Science, Faculty of Natural Sciences, Klaipeda University, Herkaus Manto 84, LT-92294 Klaipeda, *olegas.vasilecas@ik.ku.lt*

Information systems are increasingly complex, especially in the enormous growth of the volume of data, different structures, different technologies, and the evolving requirements of the users. Consequently, current applications require an enormous effort of design and development. The fast-changing requirements are the main problem in creating and/or modifying conceptual data models. To improve this process, we proposed to reuse already existing knowledge for conceptual modelling. In the paper, we have analysed reusable knowledge models. We present our method for creating conceptual models from various knowledge models.

**Keywords:** transformation, ontology, data modelling.

### 1 Introduction

Most of information systems analysts have at least once considered how many times they have to analyse the same problems, create and design the same things. Most of them ask, is it possible to reuse already existing knowledge? Our answer is yes. We believe that knowledge can and should be reused for information systems development. The knowledge reuse could be really of help to information systems analysts and engineers who develop information systems in one particular area for many years or have one software product line.

However, even these days, most domain knowledge is elicited from documents, experts and usually waste previous efforts, time, and resources. In this paper, we present available knowledge sources which could be reused in software engineering process,

---

\* The work is supported by Lithuanian State Science and Studies Foundation according to High Technology Development Program Project “Business Rules Solutions for Information Systems Development (VeTIS)” Reg. No. B-07042.

and we present a new approach of building conceptual models from these sources. Real world domain knowledge or, as we call it, domain ontology can bring outstanding benefits in software engineering. We already proved in [11] that ontologies can be reused for conceptual data modelling.

In this paper we are proposing a method of knowledge reuse for those who seek an efficient and quality driven approach for data structure development, data integration strategies, enterprise data models, logical data models, database design, data warehouse design, or data mart design.

The paper is organised as follows: the next chapter describes the theoretical background; in Chapter 3 we present and analyse available knowledge sources which could be reused for conceptual data modelling; and in Chapter 4 we present the method for knowledge reuse for data modelling.

## **2 Related Work**

In this chapter we present ontology and metamodel based transformations used in our proposed method. We also discuss quality metrics which could be used for evaluation of the method and data sources.

### **2.1 Ontology**

Computer science defines ontology as “a formal, explicit specification of a shared conceptualization” [24]. This definition is based on the idea of conceptualization: a simplified view of the world that we want to represent. Conceptualization is the process by which human mind forms an idea about part of the reality. This idea is a mental representation free of accidental properties and based on essential characteristics of the elements. Therefore, the (computer science) ontology concept is joined to a domain or mini-world, and the specification represented in ontology is concerned with that domain. In computer science, the main idea to create ontologies is to take a concrete model of the world and, through a descriptive process (function), to create an abstract model that captures its essence.

Many authors propose different ontology definitions. We accept the ontology definition proposed in [18]. Ontology defines the common terms and concepts (meanings) used to describe and represent an area of knowledge. Ontology can range in expressivity from taxonomy (knowledge with minimal hierarchy or a parent/child structure) to thesaurus (words and synonyms) to a conceptual model (with more complex knowledge), to a logical theory (with very rich, complex, consistent, and meaningful knowledge).

### **2.2 Metamodel-Based Transformations**

The notion of model transformation is central to Model Driven Engineering. A model transformation takes as input a model that conforms to a given metamodel and produces as output another model that conforms to a given metamodel. A model transformation may also have several source models and several target models. One of the characteristics of a model transformation is that a transformation is also a model, i.e. it conforms to a given metamodel. More information on metamodels can be found in [16].

Model Driven Architecture (MDA) [17] defines three viewpoints (levels of abstraction) from which a system can be viewed. From a chosen viewpoint, a representation of a given system (viewpoint model) can be defined. These models each correspond to the viewpoint with the same name, and they are Computation Independent Model (CIM), Platform Independent Model (PIM), and Platform Specific Model (PSM). MDA is based on four-layer metamodelling architecture, and several OMG's complementary standards. There are the meta-metamodel (M3) layer, metamodel (M2) layer, model (M1) layer, and instance (M0) layer.

We analyse ontology transformation to a data model. The mapping from OWL (ontology web language) to ER was described in [18]. However, this mapping is incomplete and it is not clear which elements from the OWL ontology are not transformed into data model. As a result, some information from OWL ontology cannot be used in data model. Metamodel-based transformations are shown in Figure 1.

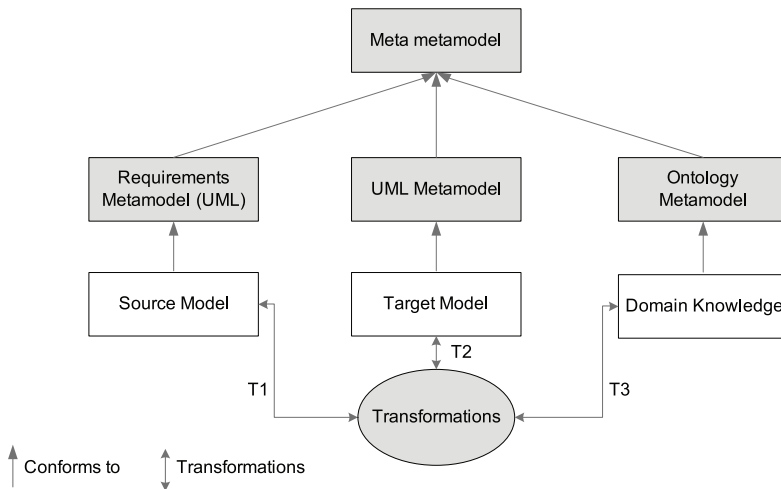


Fig. 1. Metamodel-based transformation

### 2.3 Quality Properties

We have to discuss the quality properties that could be used for transformation evaluation. The paper [19] provides a list of quality properties. The main quality properties are: annotation, appropriateness, completeness, conceptual clarity, consistency, correctness, expressiveness, testability, unambiguity, understandability, verifiability. However, this list is not very useful because we need a systematic approach to quality improvements.

Another author [20] defines the following properties.

- **Legibility.** To measure legibility which expresses the ease with which a conceptual schema can be read, we propose two subcriteria, namely clarity and minimality. Clarity is a purely aesthetic criterion. It is based on the graphical arrangement of the elements composing the schema. The second sub-criterion is minimality. A schema is said to be minimal when every aspect of the requirements appears only once.

- Expressiveness. A schema is said to be expressive when it represents users requirements in a natural way.
- Simplicity. A schema is said to be simple if it contains the minimum possible constructs.
- Correctness. The property is used in a wide range of contexts leading to very different interpretations. A schema is syntactically correct when concepts are properly defined in the schema.
- Completeness. A schema is complete when it represents all relevant features of the application domain [10]. More specifically, the completeness can be measured by the degree of coverage of user's requirements by the conceptual schema.
- Understandability. Understandability is defined as the ease with which the user can interpret the schema. This criterion is very important for the validation phase and, consequently, influences directly the measure of completeness. The understandability of a conceptual schema relies on how much modelling features are explicit.

Using these properties, we will evaluate transformation.

### **3 Available Domain Knowledge Sources**

In this section we review and analyse available knowledge sources which could be reused for conceptual data modelling. It is very important to have good quality domain knowledge source because transformation quality straightforwardly depends on knowledge quality. Of course, the transformed model can and should be improved manually.

Information systems are increasingly complex, especially because of the enormous growth of the volume of data, different structures, different technologies, and the evolving requirements of the users. Consequently, current applications require an enormous effort of design and development. In fact, such applications require a detailed study of their fields in order to define their concepts and to determine the concepts' relationships [10].

Knowledge models are reusable models, in other words, 'templates' to help jump start and/or quality assure data modelling efforts. They can be used to save time and costs on efforts to develop enterprise data models, logical data models, database, and data warehouse.

There are many books and articles written on the subject of data modelling, and most system professionals know how to model data. Actually, what they need are reusable knowledge models which could be reused for real projects and could save many hours of work [2].

If we want to get a high quality data model after transformation, the knowledge model has to be simple, correct, and complete.

If the knowledge model is not simple after the transformation, we will get a complex conceptual schema, which will have to be improved manually.

If the knowledge model is not correct, after the transformation we will get the same mistakes in the conceptual model.

If the knowledge model is not complete (at least in our domain), after the transformation we will get an incomplete conceptual model.

We propose to improve knowledge model iteratively. All mistakes noticed in the conceptual model should also be rechecked in the knowledge model. Also, if the conceptual model is incomplete, we have to add the needed information into the knowledge model. Step by step we can create a sophisticated source of knowledge. We will not discuss the creation of ontologies in this paper because this topic needs more attention and is out of the scope of this paper.

In the next chapters, we analyse different knowledge sources which could be reused for conceptual data model building, and we try to evaluate which of them is the most suitable.

The knowledge sources can be classified into three main categories – commercial (for example, IBM data model, industry data models [2]), freely available (for example, SUMO and OpenCyc), and manually created for a specific purpose.

### 3.1 SUMO

In this section we present SUMO domain ontology which could be used as a knowledge model. The main difference between universal models and domain ontologies is that usually ontologies are more abstract. To reuse ontology is more complicated than to reuse a universal model. However, domain ontologies are a really good knowledge source that could be reused.

SUMO is the largest free, formal ontology [1]. It contains more than 20,000 terms and 70,000 axioms if all domain ontologies are combined. SUMO consists of SUMO itself, the Mid-Level Ontology (MILO), and domain ontologies (communications, countries and regions, distributed computing, economy, finance, engineering components, geography, government, military, North American industrial classification system, people, physical elements, transportation, etc). SUMO is written in the SUO-KIF language.

SUMO ontology also provides check rules. Most significantly, SUMO ontology is freely available for everyone. Everyone can participate in the ontology development and improvement process. Other advantage is that ontologies provide a set of rules, i.e. we can restrict the model. However, most of these ontologies do not cover all domain areas.

### 3.2 Cyc and OpenCyc

OpenCyc [5] is the open source version of the Cyc technology, the world's largest and most complete general knowledge base and commonsense reasoning engine. The entire Cyc ontology contains hundreds of thousands of terms, along with millions of assertions that relate the terms to each other, forming an upper ontology, whose domain is all the human consensus about reality.

The Cyc project has been described as “one of the most controversial endeavours of the artificial intelligence history” [13]; hence, it has inevitably garnered its share of criticism.

OpenCyc is similar to full Cyc, but its knowledge base is just a few percent of the full knowledge base and its functionality is greatly reduced. Since Cyc's success lies in



the completeness of its knowledge base, the only people who really know the extent of Cyc's progress are Cycorp employees [4].

Furthermore, with trying to cover the entire world, Cyc becomes too enormous and abstract. The reuse of this ontology is very complicated. Some issues about Cyc reuse are discussed in [14].

### **3.3 Wikipedia-Based Ontologies**

In recent years some communities tried to extract structured information from Wikipedia. As a result, YAGO [7], DBpedia [8], and FreeBase [9] ontologies were created. In this section we shortly introduce these ontologies.

YAGO is a huge semantic knowledge base. According to [7], YAGO contains more than 2 million entities and 20 million facts about these entities. The authors of YAGO state that YAGO has a manually confirmed accuracy of 95%.

DBpedia [8] is a knowledge base which allows to make sophisticated queries in Wikipedia, and to link other data sets on the Web to Wikipedia data. The DBpedia data set currently provides information on more than 2 million entities. Altogether, the DBpedia data set consists of 218 million pieces of information (RDF triples). The accuracy of DBpedia is not confirmed at the moment.

Freebase [9] is an open, shared database that contains structured information on millions of topics in hundreds of categories. This information is compiled from open datasets like Wikipedia, MusicBrainz, the Securities and Exchange Commission, and the CIA World Fact Book, as well as contributions from user community. The accuracy of Freebase is not confirmed at the moment.

YAGO, DPpedia, and Freebase knowledge sources are really valuable. At the moment DBpedia is the biggest Wikipedia-based ontology. The accuracy of YAGO has been confirmed.

### **3.4 Industry Data Models**

The book [2] provides a series of industry universal data models for each phase of an enterprise's business cycle: people and organizations, products (services or physical goods), commitments which are established between people and/or organizations, transport shipment, work efforts, invoices, budgeting and accounting, human resources management and tracking.

An industry data model or universal data model is a model that is widely applied in some industry. Sufficiently effective industry data models have been developed in banking, insurance, pharmaceuticals and other industries to reflect the strict standards applied in customer information gathering, customer privacy, consumer safety, or "just in time" manufacturing.

The authors of the book [2] claim that 60% of a data model (corporate or logical) or data warehouse design consists of common constructs that are applicable to most enterprises. This means that most data modelling or data warehouse design efforts are at some point recreating constructs that have already been built many times before in other organizations.

The authors provide nine subject areas: accounting and budgeting, human resources, invoicing and billing, orders and agreements, people and organizations, product, shipments and deliveries, web and e-commerce, work effort and project management.

In addition, several industry specific universal data models are available, including banking, investments and financial services, healthcare, insurance, manufacturing, professional services, telecommunications, and travel.

These universal data models are very useful for data modelling. However, we can reuse only the structure; these models do not contain any business rules, which also are a very important knowledge resource.

### 3.5 Commercial Data Models

There are many commercial data models which could be reused. We analysed the well-known IBM data model M1. M1 database contains the Banking Data Warehouse Model. The model is composed of an entity-relationship model for application development. It contains 910 entities and 5237 attributes.

Let us examine the ‘product’ definition of the IBM data model.

‘Product’ identifies goods and services that can be offered, sold, or purchased by the financial institution, its competitors, and other involved parties, or in which the financial institution has an interest during the normal course of its business activity; for example, ‘Product#220 (Xyz bank’s private banking residential mortgage loan)’, ‘Product #988 (personal checking account)’, ‘Product #440 (securities trading service)’. ‘Product’ has 22 attributes and 28 relationships in the model. A small part from the model is provided in Figure 2.

The IBM data model has three levels – the abstract level, the middle level, and the model level. Such organisation of the model is very convenient for reuse.

Commercial data models usually are very expensive and there are many restrictions.

### 3.6 Other Ontologies

Protégé [6] provides more than 50 domain ontologies; however, none of them can be used for conceptual data modelling because most of them contain only a few dozens of concepts and are totally immature.

WordNet is a large lexical database of English [15]. Nouns, verbs, adjectives, and adverbs are grouped into sets of cognitive synonyms (synsets) that each express a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of meaningfully related words and concepts can be navigated with the browser. WordNet is also freely and publicly available for download. WordNet’s structure makes it a useful tool for computational linguistics and natural language processing. WordNet cannot be straightforwardly reused, but it is very useful for finding synonyms and more abstract terms.

## 4 The Proposed Approach

In the previous section we analysed available sources which could be reused for conceptual modelling. In this section we describe our method for knowledge reuse for conceptual modelling. We performed a few experiments, including reuse of knowledge from already existing resources and from the resources which were created manually.

We briefly describe the proposed method for building a conceptual model from reusable models. The method consists of four main steps.

1. Building of requirements using ontology.
2. Finding or creation of an appropriate knowledge source which can be used for transformation.
3. Transformation of the knowledge model into a specific data model with our plug in OntER. The created data model can be opened with Sybase Power Designer 12.0 tool and adapted for specific needs. Transformation rules can be found in [24].
4. The last step is the generation of the physical data model with Power Designer 12.0 for a particular database management systems (DBMS). This feature is already implemented in the original version of Power Designer 12.0.

By a simple generation procedure, the conceptual data model can be transferred to the physical data model. The physical data model adapts your design to the specifics of a DBMS and puts you well on the way to complete physical implementation.

The transformation process is shown in Figure 2.

The first experiments were carried out with the domain ontology found in Protégé site [6]. Other experiments were performed with ontology created by us. Finally, we tried to experiment with other ontologies.

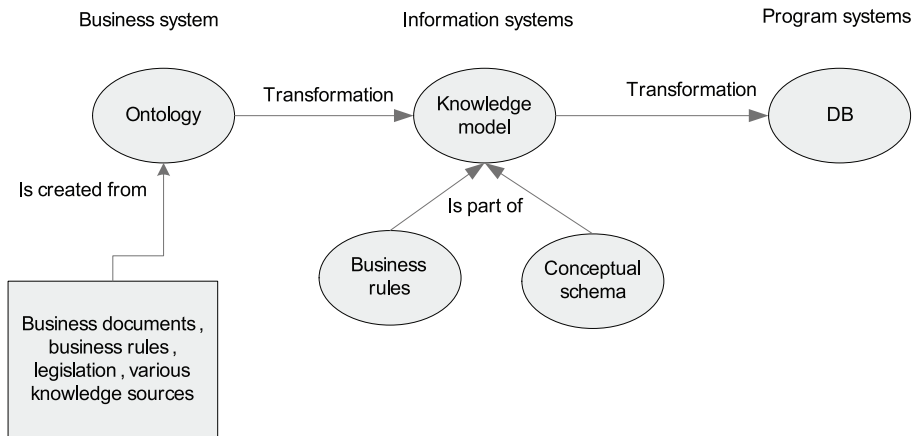


Fig. 2. The transformation process

We created a requirement metamodel with the Eclipse tool (Figure 3). We also used the requirement model taken from [22]. This requirement model [21] is composed of a Function Refinement Tree (Figure 4) to specify the hierarchical decomposition of the system, a Use Case Model to specify the system communication and functionality, and Sequence Diagrams specify the required object-interactions that are necessary to realize each Use Case. The ontology is presented in Figure 5.

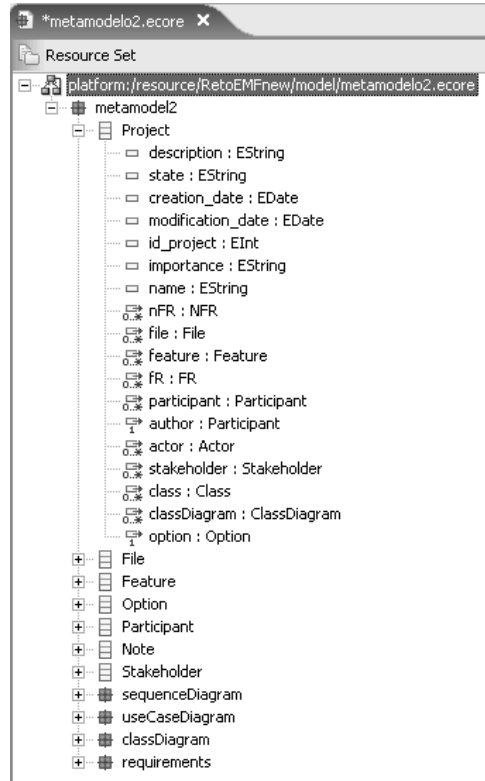


Fig. 3. Requirement metamodel

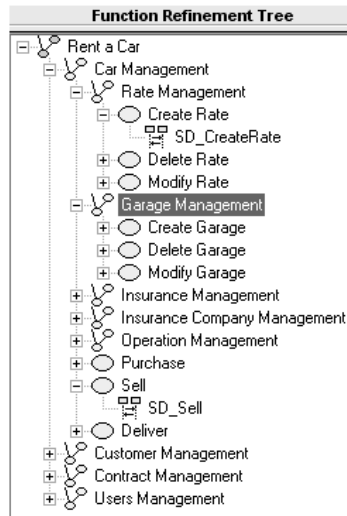


Fig. 4. Function Refinement Tree

Below a small piece of ontology “Enterprise” in Protégé (frames) format (defclass Supplier “someone whose business is to supply a particular service or commodity”) is provided.

```
(is-a Enterprise)
(role concrete)
(single-slot city
;+ (comment “a place (city) where supplier is located”)
    (type INSTANCE)
;+ (allowed-classes City)
;+ (cardinality 0 1)
    (create-accessor read-write))
(single-slot country
;+ (comment “country where supplier is located”)
    (type INSTANCE)
;+ (allowed-classes Country)
;+ (cardinality 0 1)
    (create-accessor read-write))
(single-slot name_
;+ (comment “name of supplier”)
    (type STRING)
;+ (cardinality 1 1)
    (create-accessor read-write)))
```

Below the result of “Enterprise” ontology transformation into a conceptual model in Power Designer native format is provided.

```
( <o:Entity Id="o58">
<a:ObjectID>8B135BB3-7264-4809-910F-3DD4BEFC7DE0</a:ObjectID>
<a:Name>Supplier</a:Name>
<a:Code>Supplier</a:Code>
<a:CreationDate>1144263520</a:CreationDate>
<a:Creator>Justas</a:Creator>
<a:ModificationDate>1144263684</a:ModificationDate>
<a:Modifier>Justas</a:Modifier>
<c:Attributes>
<o:EntityAttribute Id="o92">
<a:ObjectID>07DEFDAC-0AD3-4A1C-AE20-4AD63A6A065A</a:ObjectID>
<a:CreationDate>1144263539</a:CreationDate>
<a:Creator>Justas</a:Creator>
<a:ModificationDate>1144263539</a:ModificationDate>
<a:Modifier>Justas</a:Modifier>
<c:DataItem>
<o:DataItem Ref="o93"/>
</c:DataItem>
</o:EntityAttribute>
</c:Attributes>
</o:Entity>
```

Below a small piece of ontology “Salary” in Protégé (OWL) format and in graphical form (Figure 5) is provided.

```
<owl:Class rdf:ID="WageRate">  
  <rdfs:comment xml:lang="en">Wage rate is amount of money  
  paid per unit of time or per unit of products</rdfs:comment>  
  <rdfs:subClassOf>  
  <owl:Class rdf:about="#Wage"/>  
  </rdfs:subClassOf>  
  <owl:disjointWith>  
  <owl:Class rdf:ID="Quantity"/>  
  </owl:disjointWith>  
  <owl:disjointWith>  
  <owl:Class rdf:ID="Time"/>  
  </owl:disjointWith>  
</owl:Class>
```

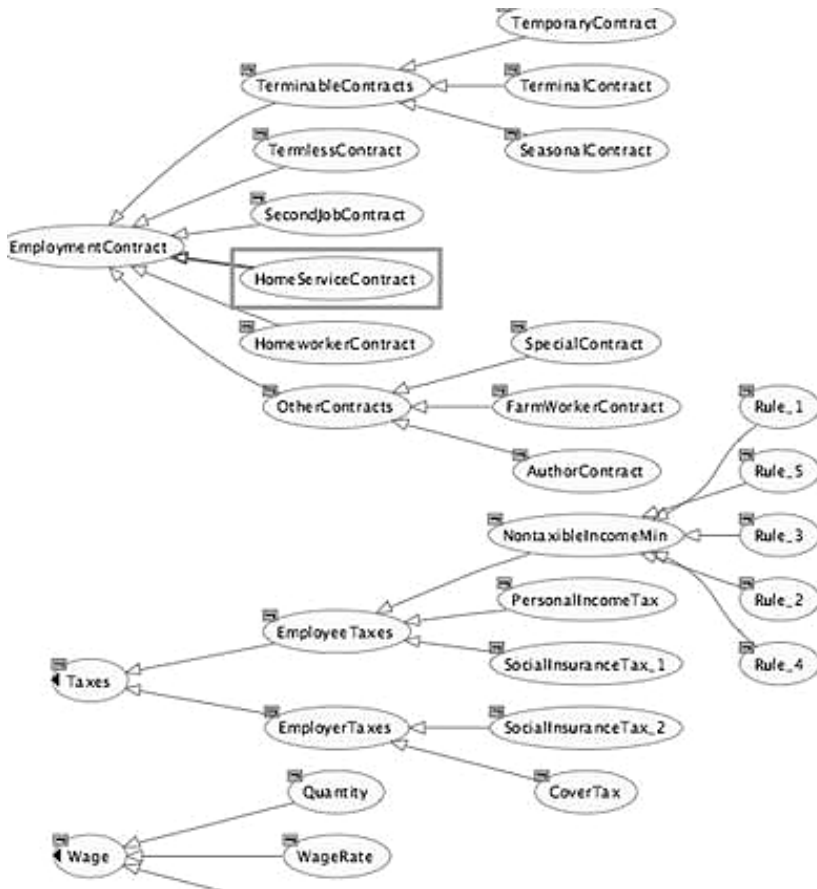


Fig. 5. Payroll ontology

Below the result of “Salary” ontology transformation into a conceptual model in Power Designer native format is provided.

```
( <o:Entity Id="o40">
  <a:ObjectID>2E0D229D-A725-4078-9EFD-D6F2F036E775</a:ObjectID>
  <a:Name>WageRate</a:Name>
  <a:Code>WageRate</a:Code>
  <a:CreationDate>1144314871</a:CreationDate>
  <a:Creator>Justas</a:Creator>
  <a:ModificationDate>1144314877</a:ModificationDate>
  <a:Modifier>Justas</a:Modifier>
```

## 5 Conclusions and Future Works

We analysed available knowledge sources which could be reused for conceptual data modeling and proposed a method for knowledge reuse. The experiment showed that the proposed method is really effective. However, it is very important to have a good enough quality domain knowledge source because evaluation of the transformation process showed that quality strongly depends on the quality of the knowledge model.

After a thorough analysis of available knowledge sources, we decided that the most completed ontology is CYC. The most expressive is SUMO. However, the most suitable by all the quality properties listed in the Chapter 2 are universal and commercial data models.

Nevertheless, after series of transformations carried out, we found that many tools do not follow already accepted standards, and this situation makes research work even more difficult. We plan to extend the research work and create OWL to UML transformation in Eclipse environment with ATL language.

## 6 References

1. Suggested Upper Merged Ontology (SUMO), <http://www.ontologyportal.org/>, [2008-04-28].
2. Silverston L. (2001) The Data Model Resource Book: A Library of Universal Data Models for All Enterprises, Revised Edition, Volume 1. John Wiley & Sons.
3. Embarcadero Technologies, <http://www.embarcadero.com/>, [2008-04-20].
4. Friedman J. The Sole Contender for AI, *Harvard Science Review* 2003, <http://www.scribd.com/doc/1814/An-Article-about-the-Cyc-Project> [2008-04-18].
5. OpenCyc 1.0.2, [www.opencyc.org](http://www.opencyc.org), [2008-04-19].
6. Protégé ontologies, <http://protege.stanford.edu/download/ontologies.html>, [2008-04-19].
7. Yago, <http://www.mpi-inf.mpg.de/~suchanek/downloads/yago/> [2008-11-03].
8. DBpedia, <http://dbpedia.org/About> [2008-11-03].
9. Freebase, <http://www.freebase.com/> [2008-11-03].
10. Mhiri M., Chabaane S., Mtibaa A., Gargouri F. *An Algorithm for Building Information System's Ontologies*. Eight International Conference on Enterprise Information Systems, Paphos, Cyprus, 2006, 467–470.
11. Trinkunas J., Bugaitė D., Vasilecas O. Formal Transformation of Domain Ontology into Conceptual Model. *Izvestia of the Belarussian Engineering Academy*, 2006, Vol. 1 (21)/2, 2006, 112–117.
12. Brewster C. et al. Data driven ontology evaluation. *Proceedings of Int. Conf. on Language Resources and Evaluation*, Lisbon, 2004.
13. Bertino E., Zarri G. P., Catania B. *Intelligent Database Systems*. Addison-Wesley Professional, 2001. ISBN 0-201-87736-8

14. Conesa J., Palol X., Olive A. Building Conceptual Schemes by Refining General Ontologies. 14th International Conference on Database and Expert Systems Applications – DEXA '06, volume 2736 of LNCS, 2003, 693–702.
15. Wordnet – a lexical database for the English language, Princeton University Cognitive Science Laboratory, <http://wordnet.princeton.edu/> [2008-04-19].
16. Kanai S., Kishinam T., Tomura T. (2000) Object-oriented Graphical Specification and Seamless Design Procedure for Manufacturing Cell Control Software Development. *Proc. of the 2000 IEEE International Conference on Robotics & Automation*, San Francisco, 401–407.
17. Miller J., Mukerji J. (eds.) (2003) MDA Guide Version 1.0. OMG Document: omg/2003-05-01. <http://www.omg.org/docs/omg/03-05-01.pdf> (2007-03-20).
18. OMG (2006) Ontology Definition Metamodel Specification. Adopted Specification 2006-10-11. <http://www.omg.org/docs/ptc/06-10-11.pdf> (2007-03-20).
19. Lindland O. I., Sindre G., Sølvsberg A. (1994) Understanding Quality in Conceptual Modeling, *IEEE Software*, v. 11 n. 2, 42–49.
20. Cherfi S., Akoka J., Comyn-Wattiau I. (2002) Conceptual Modeling Quality – From EER to UML Schemes Evaluation. In: Stefano Spaccapietra, Salvatore T. March, and Yahiko Kambayashi, editors, *Proceedings of the 21st International Conference on Conceptual Modeling*, Volume 2503 of Lecture Notes in Computer Science, Tampere, Finland. Springer-Verlag, 414–428.
21. Abrahão, S., Genero, M., Insfran, E., Carsi, J. A., Ramos, I., Piattini, M. (2008) Quality-Driven Model Transformations: From Requirements to UML Class Diagrams. In: *Model-Driven Software Development: Integrating Quality Assurance*. IGI Publishing.
22. Reto Tool, <http://reto.dsic.upv.es/reto> [2008-06-19].
23. Gruber T. R. A translation approach to portable ontology specifications. *Knowledge Acquisition* 5 (2), 1993, 199–220.
24. Trinkunas J., Vasilecas O. A Graph-Oriented Model for Ontology Transformation into Conceptual Data Model. *Information Technology and Control, Kaunas, Technologija*, 2007, Vol. 36, No. 1A, pp. 126–131.



## Using the Sponsor-User-Programmer Model to Improve the Testing Process

**Guntis Arnicans, Vineta Arnicane**

University of Latvia, Raiņa Blvd 19, Rīga, Latvia  
*Guntis.Arnicans@lu.lv, Vineta.Arnicanne@lu.lv*

This paper describes the Sponsor-User-Programmer (SUP) model, which offers simplified and structured views of certain aspects of software requirements. The principles of the model are based on the idea that there are three major players or stakeholders in software development – the client or sponsor, the user, and the programmer or software developer. Each of these people has his or her own vision and documentation about the behavior and features of the software that is being developed. The SUP model brings together software requirements, classifying them in easily perceived classes according to which stakeholders agree with them and whether they are realized. The authors give an insight into how project managers, stakeholders, and testers can use the SUP model. The relation between the SUP model and testing processes is described.

**Keywords:** requirement engineering, testing, testing process improvement, software process improvement.

### 1 Introduction

There are many methodologies for software development which offer a detailed description of the development process as such, as well as the testing process and its role in the overall design process. The availability of these methodologies, however, does not in and of itself mean that they are put to practical use. On the other hand, “[...] meeting real customer needs and improving project estimates and visibility with the customer were to be key drivers for the software process improvement” [1] in industry. Moreover, software development processes involve such people as clients and users, who are often quite unfamiliar with IT technologies and design methodologies.

In recent years, people without specific knowledge have become involved in testing procedures more and more often because of the ever more acute lack of software testing specialists. Without education and training, these employees do not ensure high-quality testing, nor are they truly aware of their role in the process at large [2]. The same is true of clients and those who finance software development. They, too, require simplified and basic knowledge to oversee the situation at hand. Hence, “modeling conventions,

methodologies, and strategies all help to simplify requirement engineering techniques so that the techniques can be used successfully by typical practitioners” [3].

The first conceptual version of the SUP model has been established to make it easier for the many persons involved in software development to understand the situation. This model, to put it simply, offers a bird’s eye view of many important aspects of the quality and testing of software. The model divides the persons involved in software development into three groups – *Sponsors* (those who commission the software, own it, and finance the process), *Users* (the target audience for the software that is being designed), and *Programmers* (those who prepare the software – programmers, system analysts, and other IT specialists). The first letters of the names of those groups form the SUP model’s name.

In the context of the SUP model, we use the term *Product* as “any deliverables of life cycle – code, documents, diagrams, etc” and term *Project* as “any piece of work we may need to test – project, on-going maintenance, emergency fix, etc”, like in [4].

The goal of introducing the SUP model is to improve the testing process by improving the understanding of all involved persons in terms of the work that is being done.

There are at least two views on quality expressed in literature [5–7]. One way of considering the quality of software is to determine whether it is in line with the requirements that have been stated. Such requirements, it is believed, are documented in a written form. That is particularly true in outsourcing projects, because documents help to uphold the formal relations between the client and the service provider.

Users, however, can have a different view of quality. They want to know whether the software meets their expectations. Software is of quality, in other words, if it is convenient and easy to use. This approach is typical in those cases when software is designed for a wide range of users, and profits depend on whether users are prepared to support the software by buying it.

The SUP model involves both of the aforementioned principles as equally important, and only the user of the model decides whether to take either or both of those principles into account. The first view of quality is based on the fact that requirements vis-à-vis the software have been documented. Thus, the SUP model includes the “*Document View*” (SUP-D). The second principle is more difficult to quantify because the understanding of quality is based on the feelings, expectations, and visions of individuals regarding the software that is being designed. This can be called the “*Vision View*” (SUP-V). The model takes into account the fact that there can be differences between an individual’s vision of the product, on the one hand, and the written requirements which have been prepared by that individual, on the other hand.

Testing activities must begin as soon as possible in the lifecycle of software development, and they must iteratively continue throughout the process [8]. This applies to the planning stage as well as to all others, and testing helps to ensure that the work that is being done is in line with all relevant stakeholders. The SUP model makes it possible to review the process at any of its stages and to determine the issues that have been inadequately addressed, as well as the issue of whether the testing has been sufficiently extensive and adequate.

The remainder of this paper is structured as follows: section 2 describes the concept of the SUP model. Section 3 demonstrates interpretation of SUP regions in detail, section 4 outlines how to use the SUP model in testing process improvement. Finally, section 5 presents conclusions and describes directions for future work.

## 2 The Concept of the SUP Model

The SUP model, in essence, is a set of various views of software under development. The main components in the model are the stakeholders—Sponsor, User, and Programmer; let us call them *Actors*—their views on the *Product*, both in terms of the Vision View and the Document View, overlap and create common areas of viewpoints. Each existing or future behavior or feature of the Product will be in line with one of these areas. The areas can be of different levels of importance in terms of how necessary for the Actors are the requirements contained in each of them. Product quality is improved when steps are taken to reduce the number of requirements in less important areas and to enhance the scope of the important areas.

### 2.1 Origins of the SUP Model

The ideas behind the SUP model have been developed over the course of several years of investigation why software testing is often inadequate and incomplete, why resources are frequently wasted in the process, and why it occurs spontaneously and without any clear planning ever so often. We determined that the main problem was that design teams have mediocre or even poor knowledge about software testing. Software sponsors and users, of course, have even poorer knowledge. Each stakeholder has a different understanding of testing, and that can lead to disagreements.

The search was on for a model that all stakeholders in software development could understand, one which would offer understandable concepts that can be described in a single presentation or lecture. The initial model was largely based on the principle described in [9]—that software behavior can be discussed from the viewpoint of specification and/or of the actual software. It is the viewpoint of software developers to software testing. Testers need to look for differences between these two sets of information. Test cases are used to test various parts of the process, both in terms of the areas of viewpoints and in terms of issues which are outside those areas. Different testing methods test software behavior in different ways. Visual images (Venn diagrams) help readers understand better what the methods do and do not test.

### 2.2 Actors in the SUP Model and Their Views

The central role in the SUP model is played by several important groups of persons involved in software development and their views on the final Product.

#### 2.2.1 Actors and Their Classification

We have defined three important groups of stakeholders in the process of software development:

- 1) the Sponsor—the person or persons who commission the software, finance its design, and wish to gain material benefits from it;
- 2) the User—the person or persons who are the target audience of the software and will use it for work or personal purposes;
- 3) the Programmer—the person or persons who design the software and receive payment or other remuneration for their work.

There are cases in which a specific individual is in several of these groups, and in that case the individual is playing several roles simultaneously. That is good in the sense that the number of different viewpoints is smaller, but it is not good in the sense that the individual’s viewpoint may be narrow and even erroneous.

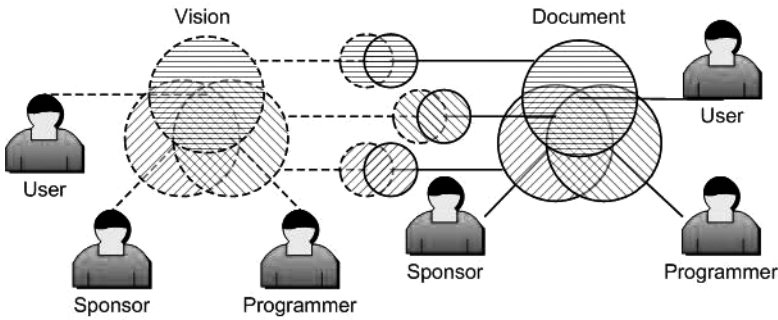





Fig. 1. Each Actor has a unique view (visions and documented requirements) of the Product. Here we see the interaction between the Sponsor, User, and Programmer.

Table 1 shows the graphic designations of the various states of software used in the diagrams of the SUP model.

Table 1

**Graphic designations of the states of software in the SUP model**

Shape	Meaning
Dashed area 	Ideas and visions about the software which are not written down
Solid line area 	Written requirements (specifications, software design, etc)
Filled area 	Requirements implemented in the software (the behavior and properties of the software, the properties of the source code, the properties of the documentation)

### 2.2.2 The Different Views of Stakeholders vis-à-vis the Product

Software and its aspects can also be reviewed at three levels:

- 1) the idea and vision as to what kind of software is needed and what its functions should be. We can regard this as the mental model of the software envisaged by an individual. Let us call this *Vision* (dashed area in the diagrams).
- 2) Visions, ideas, and requirements written down in formal documents, letters, diagrams, etc. These specify the requirements of the client, the requirements and complaints of the users, the proposed software design, etc. Let us call these *Documents* (solid line area in the diagrams).
- 3) The software which consists of the executable program, its code, and its support documentation – *Product* (filled area in the diagrams).

### 2.2.3 The Visions of the Stakeholders

Each stakeholder in software development will have his or her own ideas and visions about the behavior and properties of the software that is being designed. The closer those ideas are to the actual behavior and properties of the final Product, the more likely it is that the stakeholders will appreciate the level of its quality.

Each Actor has a different view on the necessary system behavior in most cases, and often these views are widely divergent. Figure 2 (a) shows the relationship among the various visions. Let us refer to the views of all involved Actors as the *SUP-Vision* (*SUP-V*). Each region is identified by a name – “S” for Sponsor, “U” for User, and “P” for Programmer or any combination of them. Each region here and in the rest of the paper represents graphically the appropriate set of unwritten (in *SUP-V*) or written (in *SUP-D*) requirements of software behavior and/or properties.

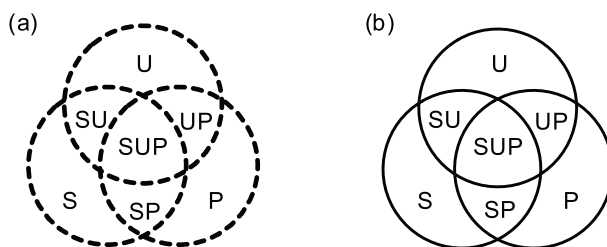


Fig. 2. The SUP-Vision View (a), the SUP-Documents View (b) and their segments

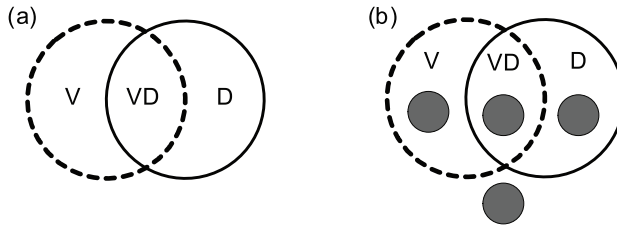
### 2.2.4 The Requirements Documented by Actors

The requirements of all Actors are identified and documented during the software development process. The developers come up not just with the source code but also, when needed, with additional specifications that are appropriate for the given situation – user handbooks, help systems, etc. We must remember that some of the documents prepared by the Sponsor and the User are also part of the final Product.

The documents can be classified in terms of their claims or requirements. Each person could look at each claim and declare whether it is or is not in line with the ideas of the relevant person vis-à-vis the software. By classifying all of the claims in the documents, we can define the view *SUP-Documents* (*SUP-D*, Figure 2 (b)). Each segment can be given an identifying name, as in the *SUP-Vision* above. It is assumed that the claim has been formally accepted by the person to whose range of interests the relevant segment belongs.

### 2.2.5 The Links between Visions and Documented Requirements

When systems are commissioned and designed, the ideas of each of the stakeholders in the process are documented. The User hands in written requirements, the Sponsor sets high-level requirements and specifications, the Programmer prepares the initial designs of the software, supplements specifications, develops the code, writes the user handbooks, etc. The fact is, however, that not all wishes are put in writing, and not all wishes are realistic. Figure 3 (a) shows this situation, V represents an area related to *SUP-Vision*, D represents an area related to *SUP-Documents*.



*Fig. 3.* The relationship between Vision and Documents and Product from the viewpoint of the User, Sponsor, or Programmer: (a) no requirements implemented; (b) some requirements from each region have already been implemented

Now let us review Figure 3 from the perspective of the User. We see that their Vision (the dashed area – V and VD) is different from the written requirements (the solid line area – VD and D). The area marked as VD is where the vision coincides with the requirements vis-à-vis the properties of the software. It is all but impossible, however, for users to put absolutely all of their desires into writing. The requirements which are not described are in the area V. There can also be mistakes in written requirements – ones that are not really in line with the User’s wishes and vision. These are in the area D. The filled areas in Figure 3 (b) represent requirements from each region that the Programmer has already implemented.

If we look at Figure 3 from the viewpoint of the Sponsor or Programmer, we see that the situation is analogous to that of the User, as described above.

In a perfect world, V and D would coincide. All the wishes of the User, Sponsor, and Producer would be written down, and that which is written down would be fully in line with that which is desired. Thus, all the wishes and ideas of the Actors would be included in the initial design, the documentation, and the later phases of the lifecycle of the software. Furthermore, everything would have been accomplished without any mistakes or misunderstandings. Of course, that never happens in real life because of the significant cost of software development and because of the fact that the various Actors inevitably have at least a few contradictory wishes.

### 2.2.6 The Interaction between the Views of Actors

As we saw in Figure 2, some of the ideas or visions are common to all Actors – what the User and Sponsor want the Programmer is ready to provide (SUP). Some ideas are common to the Sponsor and User (SU), some are common to the User and Programmer (UP), and some are common to the Sponsor and Programmer (SP). Each stakeholder usually has some idea about the software that is unknown to the other stakeholders or to which the others do not agree (“S” for Sponsors, “U” for Users, and “P” for Programmers in Figure 2). The situation with documented requirements is similar.

## 2.3 The Product and Its Relationship with Views

We have so far focused on the vision and written requirements related to the Product. The job is to produce a Product that is as close as possible to the Vision of all relevant Actors. Some of these Visions will be documented as requirements. Only

some will end up in the final software source code (in Figure 3 (b), the filled areas represent the produced software behavior and properties that are in line with the requirements in the relevant segments). The sad fact is that different people interpret requirements in different ways, which results in mistakes. This means that part of the Product will satisfy the Vision, another part will satisfy the Document (documented requirements) and the third part will satisfy both (the filled area in the segments in Figure 3). The Product, moreover, can have behavior and properties which are not in line with the desires or requirements of the Actors (the filled area outside the segments).

A few interpretations, for instance, from the perspective of the User:

- in segment VD, where visions and documented requirements coincide, we see that there are requirements which have already been implemented. They are in the filled area, and that means that not all requirements have yet been implemented, because the entire segment has not been filled in.
- If the software behavior and properties are in line with items from segment V, that means the Programmer has taken into account the User's wishes that have not been set out in writing but that are nevertheless necessary and in line with needs.
- If the behavior and properties are in line with items from segment D, that means the Programmer has implemented mistaken requirements of the User (i.e., the written document does not truly reflect the wishes of the User).
- Software code that is outside the segments V, VD, and D refers to everything else that has been done – items about which the user has neither visions nor requirements. These may be specific requirements from the Sponsor, items which simply make it easier to maintain and test the software, as well as functions nobody needs.

The (a) and (b) parts of Figure 4 together depict the structure of the SUP model, which is its basic framework for testing needs. One conclusion usually drawn by those who study the SUP model is that testing must be very diverse to check many different issues (Figure 4 (b) alone has 16 different segments – 7 segments, for instance, U, SU, UP, and 8 filled circles that represent requirements implemented in software according to each segment).

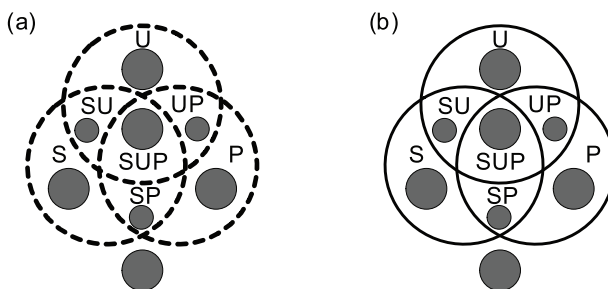


Fig. 4. SUP views during software development – (a) visions about the software (the SUP-V model) and the resulting software; (b) documented requirements for the software (the SUP-D model) and the resulting software

## 2.4 Regions of Views

### 2.4.1 Regions: the Cornerstone to Classify and Evaluate Situations

Because of the varying number of Actors, there can be various combinations in terms of situations that can be represented in the relevant region. It is not difficult to find the real-life interpretation of each region. Let us consider some of these. Various methods can be used to identify and process any situation.

Not all situations are equal in terms of necessity and utility. Furthermore, interpretation of the situation will differ depending on the current phase of the project's lifecycle. Each region in the SUP model can be weighted so as to emphasize the fact that not all regions are equal at any given time.

Let us consider an instance of the project deadline drawing near. The greatest weight will be in the filled-in field of the SUP region. That means that the views of the Sponsor, the User, and the Programmer coincide regarding the behavior and properties of the software, and the relevant requirements have been implemented in the Product. To us, this is a desirable situation.

Next, we could consider the filled areas in the SP, UP, and SU regions. These refer to those parts of software requirements which lack the support of just one of the Actors. Relationships among these regions depend on which Actor is dominant.

Unpleasant situations which are less important are represented in the open areas in the regions – the desired or required behavior has not been ensured. The worst of this situation is the open area in the SUP region at the end of the project, because there unanimity on requirements is achieved, but those requirements have not been implemented in an executable code.

A less clear situation exists with software which no one really needs – software which has perhaps been created by mistake (the filled-in field outside the regions). The open area outside the regions represents unknown future potential, both in a positive and a negative sense.

When a specific project is developed, the weighting of various regions can occur on the basis of specific principles. Within a single company, however, a weighting principle that is valid for more than one project can be developed. This is done while taking into account the company's specifics and culture.

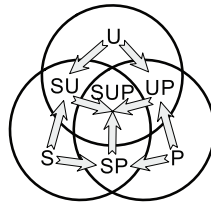


Fig. 5. Paths from regions with smaller weight to regions with greater weight

### 2.4.2 Strategies for Quality Improvements from the Perspective of Regions

Strategies for quality improvement with the help of the regions can be described in a simplified way:

- 1) identify requirements of the Product in each SUP model's region;



- 2) set a goal in terms of the target region in which we wish to see each identified requirement. The goal may be to throw out the unnecessary requirement, too;
- 3) choose the pathway from the initial region to the target region, moving to neighboring regions which are of greater weight (if there are several pathways, we choose the optimal one);
- 4) take steps to achieve our goals.

The first of the steps is primarily based on various testing methods. The fourth step can be taken by changing the Product appropriately or by using explanations and arguments to change the views of other Actors regarding the Product.

## 2.5 The Dynamics of the SUP Model

In working with the SUP model, we must remember that its components change their status over the course of time. That is primarily because people change their minds about things. They like things which they used to dislike and vice-versa. If Document View has changed, the documents become erroneous. Those who maintain the software can make changes which are no longer in line with the vision of the documentation of an Actor.

Depending on the segment that is affected, changes can be radical or minimal; positive or negative. The SUP model can help judge how important the changes are and decide whether they are, generally speaking, desirable or undesirable.

## 3 Interpretation of SUP Regions

One of the main goals of the SUP model is to inform Actors about typical situations that are described in the regions so that the software testing process can be aimed at identifying those situations; furthermore, steps can be taken to change them in the desired way.

Let us look at the situation in Figure 4 (b) (relations between SUP-D and the Product).

- **The filled-in part of the SUP region** is the ideal situation in terms of this view because it represents a set of specifications about which the Sponsor, User, and Programmer have reached a written agreement. The Product fully corresponds to those specifications. We want this area to be as large as possible. In terms of testing, it would be enough if only a couple of tests showed correspondence to this section. Additional tests ensure greater psychological security, but they really are a waste of resources. Examination of sample tests will not change the Product quality. The tests that are available in this segment are very useful, however, to explain the specifications more precisely, to conduct testing during the software development process, and to offer examples in the user documentation that is used.
- **The open area in the SUP region** represents the issues which all stakeholders have agreed upon but which have not been implemented in the code of the Product for one reason or another. One must carefully consider how the relevant properties can be ensured in the Product and what resources are needed for that purpose. As the design process draws to a close, testing must focus primarily on this region, because the existence of such region makes it clear that the Programmer has failed in one

way or another. We must remember, however, that from the perspective of SUP-V, if the relevant behavior is in a different region, the specifications must be changed. It is quite possible that there is a fundamental reason for such a situation, and that reason must be identified.

- **The filled-in area in the SU region** represents things which Users wanted and the Sponsor specified and which the Programmer did in opposition to their own official view. If the Programmer can officially accept that, the requirements of functionality move to the filled-in area of the SUP region (this is required because otherwise the Programmer can just remove functionality from the Product).
- **The open area in the SU region** represents things which Users want and have specified, but which the Programmer cannot promise and perhaps has no intention to do. If compromise can be found as the project develops, the situation might move from this region to the SUP region.
- **The UP region** – the User has written down requirements which the Programmer is prepared to accept (the open area) or has already implemented (filled-in area), but which the Sponsor is not going to finance or which are not included in project specifications for any other reason. This situation often occurs in in-house projects, when the Programmer communicates directly with the User, but specifications are never really put together because of lack of time or other reasons. What happens next depends on whether the relevant request has already been satisfied. The User and/or Programmer can attempt to convince the Sponsor to change the specifications. It is also possible that everyone is satisfied with the situation, even though it has not been put into the specification – resources have been saved! Another possibility is that a key goal of the Sponsor is not pursued precisely. For instance, the Sponsor may want to limit user access to data, while the User wants to access all data (the latter is simpler to implement for the Programmer). In that case, the developed code may have to be revised.
- **The SP region** represents things which have been listed in the specification, implemented in the software, or are to be implemented by the Programmer, but which the User does not need. The reason may be that the User is simply not interested in them. This applies to things such as the security system, testware to make testing easier, handling of mistakes, or establishment of an audit trail. It may be, however, that because of incompetence, functions that are bothersome or even hazardous for the User are requested and installed. The User and the Sponsor alike need to be convinced to reach an agreement on what to do. The Sponsor, of course, is free to spend his/her money as he/she sees fit; nonetheless, specifications can be mistaken.
- **The S region** represents things that are listed in specifications, are not demanded by the User, and which the Programmer does not intend to implement. These situations can move to the SP region as the Programmer continues their work, but it is also possible that the requirements which are found here will never be pursued because the User categorically objects to them. The response here is similar to that in the SP region.
- **The U region** represents things which the User wants but the Sponsor is, for one reason or another, unwilling to put in the specifications for the Programmer. The Users can communicate with the Sponsor to ensure that the requirements are placed in the specifications (in which case the situation moves to the SU region), or they can

deal directly with the Programmer (PU), reaching an agreement with the Sponsor afterwards. Sadly, this is quite common in real life, because the Users are often brought into the development process very lately or not at all.

- **The P region** represents behavior or properties of software that are denoted in the software development or in the documents which the Programmer has put together for their needs, but which have been requested neither by the User nor by the Sponsor in their specifications. Perhaps such functionality is unnecessary, or it is necessary but neither the Sponsor nor the User has thought of it, therefore it is not in the specification. This often applies to functionality the technical personnel that run the system need. This can concern things such as the LOG trail to find out what work or calculations are being done. Specifications usually do not include handling of low-level errors.

If sufficient real life examples are given when potential users of the SUP model are taught about the SUP model, they quickly learn to identify situations and to see what should be done in response to them.

## 4 Improvement of the Testing Process

Requirements are at heart of each software development project. Testers have to do both – validate compliance of requirements to stakeholders' claims, and verify correctness of implementation of requirements through all stages of software development.

### 4.1 Support of the Testing Process Management

Exploitation of the SUP model can in and of itself improve the testing process because structured knowledge about all possible situations make it possible to view testing in a different way and to restructure the relevant activities. The model defines directions to pursue, but it does not address testing in detail.

The SUP model can be a tool that helps start testing early, do it thoroughly and differently, guides planning of testing through the whole lifecycle of software development. All these issues are essential to quality testing [8]. In such a way SUP model supports iterative reassessment of the quality of testing itself in order to improve testing processes [4, 10, 11].

Testing allows project managers to know facts about software – what kinds of faults were found in the software, how many faults were found and where in the software they were found. Quality testing also shows what kinds of faults have been sought but not found. This is an iterative process. Each time the project is reviewed there must be consideration of whether any changes in the project have been forgotten. Thus, it is possible to reassess the entire project situation and to make informed decisions on improving the relevant processes.

The SUP model makes it possible to improve software quality by highlighting project weaknesses, such as requirements which the Programmer has followed by request of the User but which are not found in the Sponsor's specifications.

The SUP method makes it possible to use resources more rationally – not at the end of the software development process, which is when testing usually takes place. It makes it possible to learn early on that the requirements of the Sponsor (the S region) are not

in line with Users' wishes, so the Users will not accept them (in which case the situation is in the SP region). The model makes it possible to monitor whether the Sponsor has agreed upon all specifications that will affect the User before the Programmer starts the work (the desired movement is from the S region to the SU region and then to the SUP region).

## 4.2 Seeking Problems in a Project Using SUP Views

At the beginning of a project, when requirements are first being developed and the goals of the system are being designed (SUP-V view), all wishes are going to be in the S, U, and P regions. Testers can work with requirements, for instance, from the U region. They can do usability testing, reviews of these requirements in order to help the Users:

- specify their wishes;
- see which requirements are contradictory;
- prepare to advocate their requirements in the eyes of the Sponsor.

Later the SUP-D view of the model is used in testing to ensure a proper view of the project as a whole:

- the requirements on which the Actors have and have not reached an agreement;
- which requirements have been and have not been fulfilled in the software;
- recollection that there may be additional functionality that is not specifically requested.

Views of SUP model give to testers a clear sight on requirements.

- What issues is each Actor thinking about (regions S, U, P)?
- Which requirements are going to be discussed in the future (regions SU, SP, UP)?
- Which requirements are implemented in each region? Which are not? Why?
- Is there functionality in the software that is not included in the requirements? Is it really redundant or not (for instance, if requirements from UP region are implemented – issues that the User needs, the Programmer agrees on with the User, but the Sponsor is not ready to include in the official project specification)?
- Are the documented requirements truly those of the User, Sponsor and Programmer?
- Are there contradictions among the various requirements?
- Is the set of requirements complete? For instance, are there requirements with which one of the Actors is not particularly familiar, perhaps requirements in which certain Actors have no interest at all? Usability requirements which are not directly related to business requirements are often, and sadly, examples of this kind of situation. For instance, sometimes specifications state the maximum amount of time that can pass before the system responds to a user request. Seldom, however, is there an indication of the minimum font sizes on the screen – something that is important to users, given that different people have different ability of vision.

Simultaneously with the job of harmonization the Users, Sponsors, and Programmers do, testers can start to prepare use cases, scenarios, test specifications for future testing, feeling the real needs of the User and Sponsor.

### 4.3 Choice and Use of Testing Methods

While the requirements have not yet been implemented, testers use only static methods in their work – inspections, reviews, prototype usability testing on paper.

The overall lexicon and collection of requirements of the project is constantly supplemented and expanded in collaboration with other project developers.

Testers also prepare testing plans and user scenarios for those requirements which are already in the SUP region. They give thought to the dynamic testing that will be necessary in the future.

As soon as parts of the Product are implemented, testers continue the work that they have done before as well as start to inspect whether the requirements and the executable code are in line with one another.

- Have the requirements been implemented correctly?
- Have any requirements on which the Actors have not yet agreed been implemented?
- Does the code include unnecessary functionality which no one has requested?

During this period, testers use traditional dynamic test methods to make sure that all requirements have been realized implemented as intended. Implementation of unexpected requirements or of completely unnecessary functionality is identified by structural testing methods (the white box methods), exploratory testing, *ad hoc* testing, and code reviews.

Typical black box testing methods, which are based only on the written requirements of the Sponsor, are operational from the SUP-D view and in relation to all regions which involve the letter S. They have little to do with interests of the User, and they cannot be used to identify unnecessary programming in the region P or outside all of the regions.

Typical white box testing methods, by contrast, use the source code, but they cannot in and of themselves check the regions S, SU, and U because essentially they are valid for all regions which involve the letter P.

Static testing methods inspect the source code while simultaneously reviewing requirement documents. Testing occurs in all regions, but there are two key shortcomings here: testers must be highly qualified, and it is not possible to identify specific errors that affect the behavior of the software (e.g., testing the speed of reaction of the system).

Acceptance testing, which is very popular and often greatly relied upon, basically operates only in the SUP region, and that does not represent adequate testing.

## 5 Conclusions and Future Work

The SUP model described in this paper was established gradually as a means for improving the testing process in order to help address certain problems that exist in practice. The most important aspect of this model is that it offers a bird's eye view of software and its functionality from the perspective of the various Actors. The model can be used by IT specialists as well as by various businesspeople whose knowledge on IT and computers can be skimpy or incomplete.

The SUP model classifies the many situations that can emerge from the visions and written specifications of Actors and from the resulting software. When such situations are explained with understandable examples, it is easier to achieve better understanding

of the testing process among the many persons involved in a software development project. Then it is clear whether one of the Actors is being ignored, e.g., whether all potential users are or are not brought into the discussion to an adequate degree. It is also possible to tell whether testing is not too unilateral or narrowly drawn.

The User and Sponsor often do not have enough time to inspect the whole process, so a narrower model can be proposed to them – just the SUP-V or SUP-D view, with explanations of the relevant regions.

The SUP model can be used as an additional resource for better organization of software development processes throughout the lifecycle of the project. The model supports the launch of testing activities at the stage of initial visions, helping to plan the project and analyze possible risks. In later phases, the model can be used to develop a strategy for software testing, plan the testing, review and improve the plans, and select the necessary testing methods and techniques.

The SUP model makes possible an extensive testing of software in order to make sure that there is nothing distinctly unacceptable about the software development process or the Product itself.

The SUP model is used as an additional resource in real projects. Students who are trained as testers learn about the model at university and elsewhere. Up to now, the model has been used informally and intuitively. Further research is needed to formalize the SUP model and to define a more precise methodology for its use. Several algorithms or scenarios should be prepared for the use of the model, and guidelines for their use should also be created. Particular attention could be given to the issue of how to determine the existing condition of a project best – by identifying the problematic regions in the SUP model, by highlighting critical problems or weaknesses, and by clearly understanding what must be done to improve the situation. This can be achieved by analyzing all project-related activities – meetings, the involved people, the drafted documents, etc. Documents can be inspected, and everyone involved in the project can be surveyed.

## Acknowledgements

This research is partly supported by the European Social Fund with the grant “Doctoral student research and post-doctoral research support for the University of Latvia”.

## References

1. O'Hara F. (2000) European Experiences with Software Process Improvement. Proceedings of the 22nd International Conference on Software Engineering, Limerick, 635–640.
2. Arnicane V. (2007) Use of Non-IT Testers in Software Development. In: Münch J., Abrahamsson P. (eds.) *Product Focused Software Process Improvement. Lecture Notes in Computer Science*, Vol. 4589. Berlin-Heidelberg: Springer-Verlag, 175–187.
3. Cheng B. H. C., Atlee J. M. (2007) Research Directions in Requirements Engineering. *Future of Software Engineering* (FOSE '07), IEEE Computer Society, 285–303.
4. Veenendaal E. (2002) *The Testing Practitioner*. UTN Publishers, Den Bosch.
5. Lewis W. E. (2005) *Software Testing and Continuous Quality Improvement*. 2<sup>nd</sup> edn. Boca Raton, London, New York, Washington, D.C.: Auerbach Publications.
6. Conradi R., Fuggetta A. Improving Software Process Improvement. *IEEE Software*, Vol. 17, No. 4, IEEE Computer Society, July/Aug. 2000, 76–78.

7. Van Solingen R. (2000) *Product Focused Software Process Improvement. SPI in the Embedded Software Domain*. Eindhoven University of Technology, Eindhoven.
8. Perry W. E. (2000) *Effective Methods for Software Testing*. 2nd ed. New York, Chichester, Weinheim, Brisbane, Singapore, Toronto: John Wiley & Sons, Inc.
9. Jorgensen P. C. (2002) *Software Testing: A Craftman's Approach*. Boca Raton, London, New York, Washington, D.C.: CRC Press.
10. Palyagar B., Moisiadis F. (2006) Validating Requirements Engineering Process Improvements – a Case Study. First International Workshop on Requirements Engineering Visualization (REV'06 – RE'06 Workshop), 9.
11. Huo M., Zhang H., Jeffery R. (2006) An Exploratory Study of Process Enactment as Input to Software Process Improvement. Proceedings of the 2006 International Workshop on Software Quality. Shanghai, 39–44.

## Complexity of Equivalence Class and Boundary Value Testing Methods

**Vineta Arnicane**

University of Latvia, Raiņa Blvd 19, Rīga, Latvia  
*vineta.arnicane@lu.lv*

There are two groups of domain testing methods – equivalence class testing (ECT) methods and boundary value testing (BVT) methods reviewed in this paper. This paper surveys 17 domain testing methods applicable to domains of independent input parameters of a program. This survey describes the basic algorithms used by domain testing methods for test case generation. The paper focuses on the theoretical bounds of the size of test suites or the complexity of domain testing methods. This paper also includes a subsumption hierarchy that attempts to relate various coverage criteria associated with the identified domain testing methods.

**Keywords:** software testing, domain testing, equivalence class testing, boundary value testing, equivalence partitioning, partition analysis, boundary value analysis.

### 1 Introduction

A domain of a program with mutually independent parameters is a set of all combinations of all values of these parameters. The input domain can be very big. The main goal of domain testing methods is to achieve a test suite the size of which is considerably smaller than the count of all inputs of the program, and which effectively reveals failures of the program as much as possible.

The main approach is to divide the test object's input domain in subdomains or equivalence classes so that inputs from the same subdomain are processed in the same way, that is, cease the same type of program's output or behavior [1, 2]. Any input represents all inputs of the subdomain to which it belongs. Assumption of the approach is that if the few good representatives of the class are tested, most or all of the bugs that could be found by testing every member of the class are found [1–5], and if representatives do not catch the bug, the other elements of class will not either [3–5].

The equivalence classes of the program's input domain can be obtained using both the program's structural analysis – path analysis, and functional analysis – equivalence partitioning, domain analysis, boundary value analysis [3].

Path analysis approach is known since the 70s of the 20<sup>th</sup> century. It assumes that domain error occurs if an input traverses the wrong path through a program [6]. Symbolic



execution can be used to obtain path constraints [7] or system of inequalities [6, 8–10], solution of which determines the subdomain of input domain, ensuring execution of this path.

Partition analysis technique has developed since the 80s of the 20<sup>th</sup> century [11–15]. Equivalence partitioning technique subdivides input domain in equivalence classes on the basis of the program's specification [1–5]. The first step is to determine the domain of valid values for each of the program's parameters. These are the equivalence classes of all valid input values, one class for each parameter. Second step is to establish the domain of all invalid input values for each parameter. Invalid values are values that are possible for the input parameter but are not included into the class of valid values. By subsequent steps, the equivalence classes are refined in such a way that each different requirement and each different program behavior is a response to an equivalence class.

The same principle of dividing into equivalence classes can be used for output data, too. As a result, the equivalence classes of input values are refined according to the principle that if two different inputs of program cause outputs that belong to different equivalence classes of outputs, these inputs should be in different equivalence classes of input values, too [1, 16].

Boundary value analysis devotes special attention to boundaries of equivalence classes, because praxis shows that boundary values often reveal faults. Boundary value testing methods for test cases choose boundary values, special values, as well as values that are close to them – just above them or just below them.

There is the group of methods that different authors call domain testing methods or domain analysis methods [17]. These methods also take into account dependencies or interactions between input parameters [3, 4, 17–20]. By these methods, the input domain often is seen as a geometrical shape and its edges – as boundaries. In most cases the domains with linear boundaries can be examined [3, 17–21], but there are some methods that allow to test nonlinear boundaries, too [4, 21–23].

Using path analysis and equivalence partitioning, equivalence classes are determined. From each class one value can be chosen as a representative of the class for testing. When we have some parameters of the program with some equivalence classes for each of them, we should use a strategy how to combine them in test cases. A similar situation is with values obtained by boundary value analysis. These approaches to domain testing methods in this paper are accordingly called equivalence class testing (ECT) methods and boundary value testing (BVT) methods.

The complexity of domain testing methods is the smallest size of the test suite generated by a method. There are two parameters for comparing the testing methods – relative effectiveness and cost [25]. The complexity of the method is closely related with the cost of the method. If complexity is high, it means that there might be a large count of test cases in the test suite. As the size of the test suite grows, more resources are necessary for testing, for instance, testers, hardware, software, time, budget. Secondly, we should consider the effectiveness of the method. A method is considered effective if the software tested thoroughly according to that method is almost correct [25]. But method should be efficient too – if the test case fails, it is better if there is a small count of candidates (input values of the test case) to blame.

Hence, it is very important to take into account the method's complexity and effectiveness during the planning stages of testing.

The aim of this paper is to review some ECT and BVT methods mentioned in literature and assess their complexity. The paper focuses on the theoretical bounds of the size of test suites or the complexity of domain testing methods. This paper also includes a subsumption hierarchy of the identified domain testing methods.

Section 2 gives some definitions useful in the context of this paper, section 3 explains the testing criteria of domain testing methods. There is a review of equivalence class testing methods in section 4 and of boundary value testing methods in section 5. In sections 4 and 5, the algorithm of testcase generation is shortly described and the complexity of each identified domain testing method is assessed. Section 6 summarizes the complexity of domain testing methods and provides the hierarchy of domain testing methods according to subsume ordering. Finally, section 7 concludes this survey with a summary of most important results and some future directions of research.

## 2 Definitions

Suppose that  $P$  is a program with  $N$  input parameters  $X_i$ , where  $1 \leq i \leq N$ . For each parameter, input domain  $D_i$  is partitioned into  $M_i$  equivalence classes with extreme points as boundary values.

The meaning of boundary value testing is to examine the program when input parameters assume extreme values for each equivalence class (maximal, minimal), just above (for some small value  $\varepsilon$ ) or just below the extreme values, and when value is *nominal* – inside the equivalence class in distance from extreme values that is considerably bigger than  $\varepsilon$ .

For the corresponding domain  $D_i$  of each parameter's  $X_i$ , each equivalence class  $d_{ij}$  of the ordered elements can be graphically represented as showed in Fig. 1. The minimal boundary value of class is  $x_{ij \min}$ , maximal boundary value is  $x_{ij \max}$ , where  $1 \leq j \leq M_i$ . Nominal value of the class is  $x_{ij \text{nom}}$ . Values  $x_{ij \min-}$ ,  $x_{ij \max-}$  are a little smaller than appropriate boundary values, but  $x_{ij \min+}$ ,  $x_{ij \max+}$  are a little bit bigger. For the sake of simplicity, *min-*, *min*, *min+*, *nom*, *max-*, *max*, *max+* instead of  $x_{ij \min-}$ ,  $x_{ij \min}$ ,  $x_{ij \min+}$ ,  $x_{ij \text{nom}}$ ,  $x_{ij \max-}$ ,  $x_{ij \max}$ ,  $x_{ij \max+}$  will be used in this paper in cases when it cannot cause misunderstanding.

Boundary values  $x_{ij \min}$  and  $x_{ij \max}$  may belong to an equivalence class, but they can also be excluded from it. Nevertheless, they are boundary values for this class.

The following inequalities hold for each  $i, j$ , when  $1 \leq i \leq N$  and  $1 \leq j \leq M_i$ .

$$\begin{array}{lll} x_{ij \min} - x_{ij \min-} \leq \varepsilon_{ij} & x_{ij \min+} - x_{ij \min} \leq \varepsilon_{ij} & x_{ij \text{nom}} - x_{ij \min} \geq \varepsilon_{ij} \\ x_{ij \max+} - x_{ij \max} \leq \varepsilon_{ij} & x_{ij \max} - x_{ij \max-} \leq \varepsilon_{ij} & x_{ij \max} - x_{ij \text{nom}} \geq \varepsilon_{ij} \end{array}$$

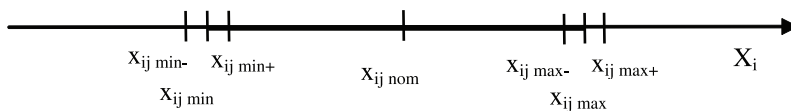


Fig. 1. Equivalence class  $d_{ij}$ :  $[x_{ij \min}, x_{ij \max}]$ , its boundary values  $x_{ij \min}$ ,  $x_{ij \max}$ , inner OFF points  $x_{ij \min+}$ ,  $x_{ij \max-}$ , outer OFF points  $x_{ij \min-}$ ,  $x_{ij \max+}$  and nominal value  $x_{ij \text{nom}}$

Let us call values just above the minimal value and just below the maximal value *inner OFF* points (they are inside equivalence class) and values just below the minimal value and just above the maximal value *outer OFF* points.

We have two assumptions:

- 1) For each parameter  $X_i$ , conjunction of all equivalence classes is domain  $D_i$ .

$$\text{So, } D_i = \bigcup_{j=1}^{M_i} d_{ij} \text{ for } \forall i, j,$$

where  $1 \leq i \leq N$  and  $1 \leq j \leq M_i$ .

- 2) There are no common values between equivalence classes –  $\forall i, j, k$ , where  $1 \leq i \leq N, 1 \leq j \leq M_i, 1 \leq k \leq M_i$  and  $j \neq k$   $d_{ij} \cap d_{ik} = \emptyset$ .

Although for each parameter, equivalence classes do not have common values, they can have common boundary values. For instance, as it is shown in Fig. 2, if domain for  $X_i$  consists of a closed interval  $[x_{ia}, x_{ib}]$  and left-open, right-closed interval  $(x_{ib}, x_{ic}]$ , they have a common boundary value  $x_{ib}$ . Let us denote the size of the set of all common boundary values for parameter  $X_i$  with  $L_i$ .

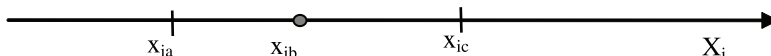


Fig. 2. Common boundary value  $x_{ib}$  for intervals  $[x_{ia}, x_{ib}]$  and  $(x_{ib}, x_{ic}]$ ,  $L_i = \{x_{ib}\}$ .

For each parameter  $X_i$ , values which belong to its domain  $D_i$  are called *valid* values, all other values – *invalid*. Invalid values can also be divided into equivalence classes. The size of the set of equivalence classes of invalid values for parameter  $X_i$  in this paper is denoted by  $Q_i$ .

Boundary value testing methods are also applicable in cases when the parameter’s domain is a set of discrete elements. It is important in such cases that there exists any function according to which elements of the domain can be ordered, for instance, months Jan, Feb, Mar, [...] Dec, lexicographical ordering of strings, or order of data in drop-down control.

### 3 Criteria of Domain Testing Methods

*Complexity of a domain testing method* is the smallest possible size of the test suite generated by the method.

The essential part of any testing method is adequacy criterion. *Adequacy criterion* explicitly specifies test case selection, determines whether a test set is adequate, and determines observations that should be done during the testing process [26].

Adequacy criteria of domain testing methods can be characterized by three aspects:

- 1) which kind of values to choose for testing, for instance, only boundary values or only representants of equivalence classes of valid values;
- 2) data coverage principle;

- 3) strategy how the chosen values are combined in test cases according to the data coverage principle.

First aspect considers semantical information of test data according to observations of the testing method, for instance, only boundary values are used or OFF points are added, too, whether the invalid values of parameters are examined or not.

Second aspect is combinatorial strategy based on data coverage. Simplest coverage criterion, each-used, does not take into account how selected values of different parameters are combined, while the more complex coverage criteria, such as pair-wise coverage, are concerned with combinations of values of different parameters.

*Each-used* (also known as 1-wise) coverage requires that every selected value of each parameter is used at least in one test case of the generated test suite [27].

*Pair-wise* (2-wise) coverage requires that every possible pair selected values of different parameters are included in the test cases of the test suite.

*T-wise* coverage requires that every possible combination of values of  $t$  parameters is included in the test cases of the test suite. N-wise coverage is a special case of *t-wise* coverage where N is the number of parameters.

As coverage degree  $t$  grows, the size of the test suite generated by the testing method also grows. Each-used and N-wise coverage are widely applied in domain testing methods.

Third aspect concerns the strategy how combinatorial coverage of chosen data is implemented. For instance, each-used coverage can be achieved in different ways:

- 1) for every chosen boundary value of each parameter, generate its own test case where all other parameters assume nominal values. In this case the size of the test suite will be the sum of the count of selected values for all parameters together;
- 2) all parameters assume boundary values in each test case – the size of the test suite will be equal to the count of selected values of that parameter for which this count is maximal.

The advantage of the second strategy is the smaller size of the test suite; however, if the test case fails, it is very hard for the tester to say why. The advantage of the first strategy is the possibility to suspect of processing of used boundary value because all other parameters assume nominal values. It is called single fault assumption, which states that faults are very rarely the result of the simultaneous occurrence of two or more faults [5].

Two fault assumption means that in test cases two of input variables assume their boundary values while all remaining variables – their nominal values.

Generally, we can speak about N-fault assumption or multiple fault assumption when the method requires a boundary value for all of program's input variables for the test case.

If the test suite is generated according to single fault assumption and some test case fails, there is a good reason to suppose that the input parameter with boundary value is incorrectly processed. But such test suite will be bigger or more complex than a test suite generated according to N-fault assumption. On the other hand, if the test case generated according to N-fault assumption fails, it is a more complex task to say which boundary or boundaries were processed incorrectly.

### 4 Complexity of Equivalence Class Testing

Let us examine domain testing methods described in related works. There are two groups of domain testing methods – equivalence class testing methods and boundary value testing methods. If in related works only the methods how to pick values for tests are described, but there are no rules given regarding combination of the selected values in test cases, the author of this paper proposes that a test case is derived for each of the selected values and the other input parameters of the test case assume some valid nominal value.

For the sake of comprehensible drawings, the discussion relates to a program Z with two input parameters  $X_1$  and  $X_2$ . Parameter's  $X_1$  domain is interval  $[x_{1a}, x_{1d}]$ , which is divided into three equivalence classes  $[x_{1a}, x_{1b})$ ,  $[x_{1b}, x_{1c})$ , and  $[x_{1c}, x_{1d}]$ . Parameters'  $X_2$  domain is interval  $[x_{2u}, x_{2z}]$ , which is divided into two equivalence classes  $[x_{2u}, x_{2v})$  and  $[x_{2v}, x_{2z}]$ . There are invalid values outside domain equivalence classes. For instance, for parameter  $X_1$  such classes are  $(\infty, x_{1a})$  and  $(x_{1d}, \infty)$ , similarly, for  $X_2 - (\infty, x_{2u})$  and  $(x_{2z}, \infty)$ .

If the infinite values are boundaries for an equivalence class of valid values, they should be treated as finite – maximal or minimal values that the computer can use for an appropriate data type. The equivalence classes of incorrect values have no boundary values [1].

#### Weak Equivalence Class Testing

The weak equivalence class testing method examines one representant from each equivalence class of valid values of each parameter [2, 5, 28–32, 42]. The basis of the method is single fault assumption. It is assumed that it is enough to test once each equivalence class.

For our sample program Z, weak equivalence class testing method generates three test cases, for instance as shown in Fig. 3 (a), because the domain of parameter  $X_1$  has the biggest count of equivalence classes – 3.

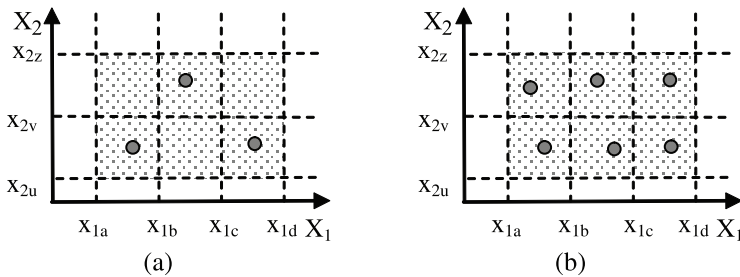


Fig. 3. Test cases for (a) weak equivalence class testing and (b) strong equivalence class testing

In general, for the program with N parameters, the size of the test suite according to the weak equivalence class testing method is  $\max_{i=1}^N (M_i)$ .

### Strong Equivalence Class Testing

The strong equivalence class testing method [5, 33] is based on multiple fault assumption. The test case from each element of the Cartesian product of equivalence classes is included in the test suite, for instance, as shown in Fig. 3 (b).

The strong equivalence class testing method allows to reach two aims – cover all equivalence classes and examine all combinations of different inputs.

In general case, the size of the test suite is  $\prod_{i=1}^N M_i$ .

### Robust Weak Equivalence Class Testing

Robust weak equivalence class testing is similar to weak equivalence testing. In addition, it also considers equivalence classes of invalid values [1, 2, 5, 28–31, 34–38, 42] according to the following algorithm:

- 1) for valid values choose only one value from each equivalence class; furthermore, all parameters have valid values in each test case;
- 2) for invalid values choose one value from each equivalence class and in each test case combine one invalid value with all other valid values. Invalid values for two or more input parameters of the program in the same test case are not allowed (see Fig. 4 (a)).

In N parameters' case, the count of generated test cases is  $\max_{i=1}^N (M_i) + \sum_{i=1}^N Q_i$ , where

$Q_i$  is the size of the set of equivalence classes of invalid values for parameter  $X_i$ .

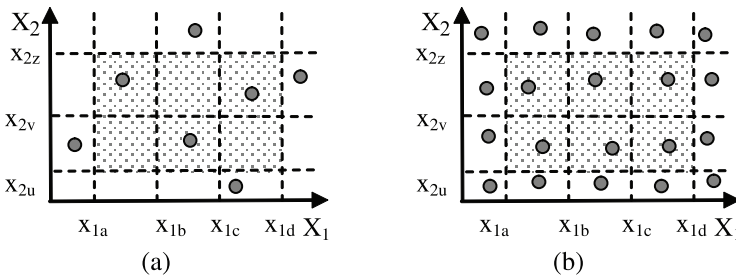


Fig. 4. Test cases for (a) robust weak equivalence class testing and (b) robust strong equivalence class testing

### Robust Strong Equivalence Class Testing

The robust strong equivalence class testing [5, 36, 39] method includes in the test suite a test case from each element of Cartesian product of all equivalence classes of valid and invalid values of all parameters, like it is shown in Fig. 4 (b).

In N parameters' case, the count of generated testcases is  $\prod_{i=1}^N (M_i + Q_i)$ .

### Robust Mixed Equivalence Class Testing

Robust mixed equivalence class testing [1] method includes in the test suite a test case from each element of Cartesian product of all equivalence classes of valid values. For invalid values, choose one value from each equivalence class and in each test case

combine one invalid value with all other valid values. Invalid values for two or more input parameters of the program in the same test case are not allowed (see Fig. 5).

In N parameters' case, the count of generated testcases is  $\prod_{i=1}^N M_i + \sum_{i=1}^N Q_i$ .

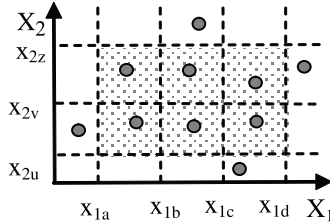


Fig. 5. Test cases for robust mixed equivalence class testing

### 5 Complexity of Boundary Value Testing Methods

#### Weak IN Boundary Value Testing

The weak IN boundary value testing method [1, 5, 33, 35] examines boundary values of equivalence classes, inner OFF points, and nominal values. This method is based on single fault assumption – one of the parameters assumes examinable values while all other parameters assume nominal values, like it is shown in Fig. 6 (a) for a 2-dimensional case.

Let us calculate the complexity of the method.

If the program has only one parameter  $X_1$ , 5 test cases are obtained for each equivalence class of valid values – min, min+, nom, max-, max. Because the count of equivalence classes is  $M_1$ , the count of test cases in the test suite is  $5M_1$ .

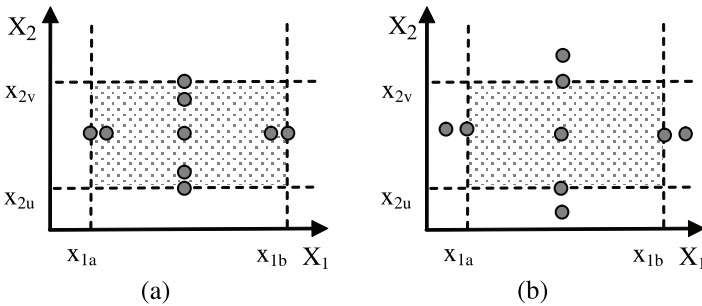


Fig. 6. Test cases for (a) weak IN boundary value testing and (b) weak OUT boundary value testing

If the program has two parameters, test cases are generated according to the following algorithm.

- 1) Hold nominal value of the first equivalence class for the first parameter and obtain 4 test cases by changing the min, min+, max-, max points for the first class of the second parameter.

2) repeat step 1 for each equivalence class of  $X_2$ .

There are  $4M_2$  test cases obtained so far.

3) Optimize the test suite – exclude redundant test cases raised by overlapped boundary values of adjacent equivalence classes.

Now we have  $4M_2 - L_2$  test cases in our test suite.

4) Repeat steps 1–3 for each equivalence class of parameter  $X_1$ .

There are  $(4M_2 - L_2)M_1$  test cases obtained so far.

5) Repeat steps 1–4 with parameters in exchanged roles.

There are  $(4M_1 - L_1)M_2 + (4M_2 - L_2)M_1$  testcases obtained during steps 1–5.

6) Now we have to add point test cases when both parameters have nominal values for all elements of Cartesian product of valid equivalence classes of both parameters.

So, we obtain  $M_1M_2$  test cases in this step.

The size of the resulting test suite is:

$$(4M_1 - L_1)M_2 + (4M_2 - L_2)M_1 + M_1M_2 = 9M_1M_2 - L_1M_2 - L_2M_1.$$

There will be  $(4N+1)\prod_{i=1}^N M_i - \sum_{i=1}^N (L_i \prod_{\substack{j=1 \\ j \neq i}}^N M_j)$  test cases in the test suite generated

by the weak IN boundary testing method for the program with  $N$  parameters. This is a lower bound of complexity of the weak IN boundary testing method.

If we skip step 3, we obtain that weak IN boundary testing will give no more than test cases. It is the upper bound of complexity of the weak IN boundary testing method, which is achievable in cases when there are no overlapped boundary values between equivalence classes.

### Weak OUT Boundary Value Testing

The weak OUT boundary value testing method examines boundary values of equivalence classes, outer OFF points, and nominal case as shown in Fig. 6 (b) [30, 34]. It is very similar to the weak IN boundary value testing method. The only difference is that the weak IN boundary value testing method uses inner OFF points while the weak OUT boundary value testing method uses outer OFF points.

The size of the generated test suite is the same as for the weak IN boundary value testing method – there will be no less than  $(4N+1)\prod_{i=1}^N M_i - \sum_{i=1}^N (L_i \prod_{\substack{j=1 \\ j \neq i}}^N M_j)$  test cases and no more than  $(4N+1)\prod_{i=1}^N M_i$  test cases.

### Weak Simple OUT Boundary Value Testing

The weak simple OUT boundary value testing method examines boundary values of equivalence classes and outer OFF points [2, 30, 31, 33, 34, 40]. The only difference from the weak OUT boundary value testing method is that it uses nominal points while the weak simple OUT boundary value testing method does not (Fig. 7 (a)).



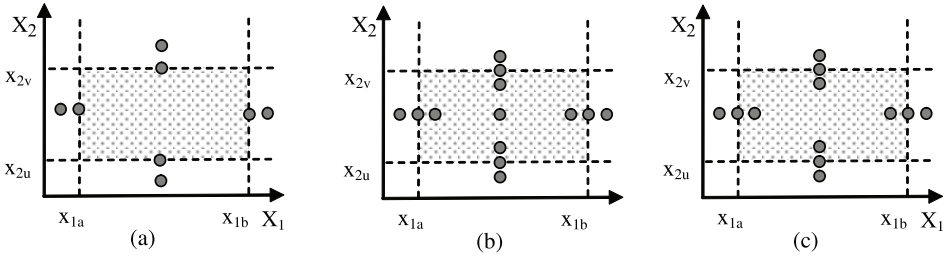


Fig. 7. Test cases for (a) weak simple OUT boundary testing, (b) robust weak boundary value testing and (c) robust weak simple boundary value testing

The size of the generated test suite is smaller than for the weak IN boundary value testing method or the weak OUT boundary value testing method – the lower bound of complexity is  $4N \prod_{i=1}^N M_i - \sum_{i=1}^N (L_i \prod_{\substack{j=1 \\ j \neq i}}^N M_j)$  test cases while the upper bound –  $4N \prod_{i=1}^N M_i$  test cases.

**Robust Weak Boundary Value Testing**

The robust weak boundary value testing method [2, 5, 34–36, 38, 42] examines boundary values of equivalence classes, inner and outer OFF points, nominal case (Fig. 7 (b)).

The size of the generated test suite can be obtained similarly to the weak IN boundary value testing method – there will be no less than  $(6N + 1) \prod_{i=1}^N M_i - \sum_{i=1}^N (L_i \prod_{\substack{j=1 \\ j \neq i}}^N M_j)$  test cases and no more than  $(6N + 1) \prod_{i=1}^N M_i$  test cases.

**Robust Weak Simple Boundary Value Testing**

The robust weak simple boundary value testing method [2, 42] examines boundary values of equivalence classes, inner and outer OFF points, but unlike the robust weak boundary value testing, it does not test nominal case (Fig. 7 (c)).

The size of the generated test suite can be obtained similarly to the weak IN boundary value testing method – there will be no less than  $6N \prod_{i=1}^N M_i - \sum_{i=1}^N (L_i \prod_{\substack{j=1 \\ j \neq i}}^N M_j)$  test cases and no more than  $6N \prod_{i=1}^N M_i$  test cases.

**Worst Case Boundary Value Testing**

The worst case boundary value testing method [5, 41] tests boundary values, inner OFF points, and nominal point. While weak IN boundary value testing used single fault assumption only, the worst case boundary value testing method also uses multiple fault assumption. It means that the method checks what happens if one or all parameters of the program assume special values (Fig. 8 (a)).

If the program has only one parameter  $X_1$ , we obtain 5 test cases for each equivalence class. After that redundant test cases raised by overlapped boundary values of adjacent equivalence classes should be excluded.

So, there are  $5M_1 - L_1$  test cases in one parameter's case.

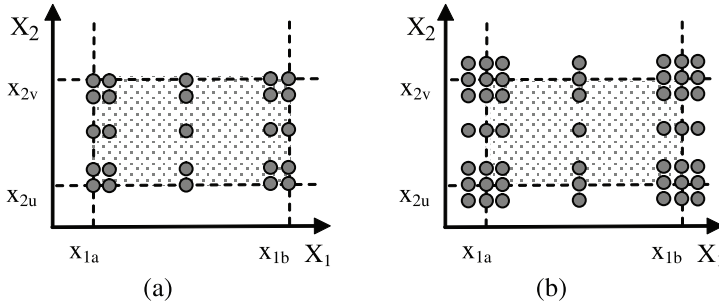


Fig. 8. Test cases for (a) worst case boundary testing and (b) robust worst case boundary testing

If the program has two parameters, test cases are generated according to the following algorithm.

- 1) Hold minimal value of the first equivalence class for the first parameter and obtain 5 test cases by changing the min, min+, max, nom, max+ points for the first class of the second parameter.
- 2) Repeat step 1 for min+, max, nom, max+ of the first equivalence class of  $X_1$ . There are  $5 \times 5$  test cases obtained so far.
- 3) Repeat the steps 1–2 for each equivalence class of  $X_1$  and optimize the test suite by excluding redundant test cases.

Now we have  $5 \times (5M_1 - L_1)$  test cases in our test suite.

- 4) Repeat steps 1–3 for each equivalence class of parameter  $X_2$ .

There are  $(5M_1 - L_1) \times (5M_2 - L_2)$  test cases obtained so far.

For  $N$  parameters' case, we obtain  $\prod_{i=1}^N (5M_i - L_i)$  test cases as the method's lower bound of complexity, but as the upper bound of complexity, the worst case boundary value testing method will give no more than  $\prod_{i=1}^N 5M_i = 5^N \prod_{i=1}^N M_i$  test cases.

### Robust Worst Case Boundary Value Testing

The robust worst case boundary value testing method [5, 39] tests boundary values, inner and outer OFF points, and nominal point (Fig. 8 (b)).

Acting according to the algorithm that is analogical to the worst case boundary value testing method, we obtain that program with  $N$  parameters will have  $\prod_{i=1}^N (7M_i - 2L_i)$  test cases or no more than  $7^N \prod_{i=1}^N M_i$  test cases.

**Weak Corner IN Boundary Value Testing**

Weak corner IN boundary value testing complies with multiple fault assumption. It tests cases when all parameters assume the same type of special values – boundary values, inner OFF points, or nominal point as it is shown in Fig. 9 (a) [1, 41].

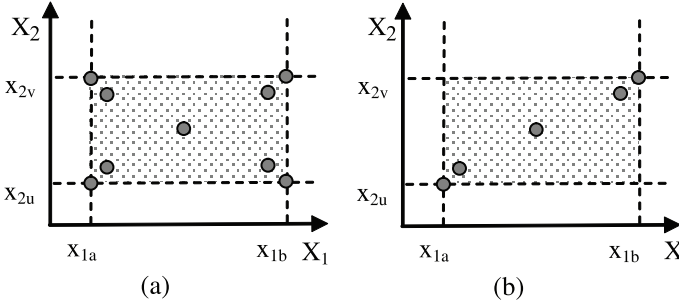


Fig. 9. Test cases for (a) weak corner IN boundary value testing and (b) weak diagonal IN boundary value testing

If the program has two parameters, test cases are generated according to the following algorithm.

- 1) Let us take boundary values of the first equivalence class of both parameters. We obtain 4 test cases (min, min), (min, max), (max, min), and (max, max).
- 2) Let us consider inner OFF points of the first equivalence class of both parameters. We again obtain 4 test cases (min+, min+), (min+, max-), (max-, min+), and (max-, max-).
- 3) Nominal points of the first equivalence class of both parameters will give one test case (nom, nom).

Now we have  $2 \times 2^2 + 1$  testcases for the first element in the set of Cartesian product of equivalence classes. For all elements, there will be  $(2 \times 2^2 + 1) M_1 M_2$  test cases.

- 4) Now exclude redundant test cases raised by overlapping boundary values and obtain  $(2^{2+1} + 1) \prod_{i=1}^2 M_i - 2 \sum_{i=1}^2 (L_i \prod_{\substack{j=1 \\ j \neq i}}^2 M_j)$  test cases.

There will be  $(2^{N+1} + 1) \prod_{i=1}^N M_i - 2 \sum_{i=1}^N (L_i \prod_{\substack{j=1 \\ j \neq i}}^N M_j)$  test cases in the test suite generated

by the weak corner IN boundary testing method for the program with N parameters.

If we skip step 4, we obtain that the weak corner IN boundary value testing method will give no more than  $(2^{N+1} + 1) \prod_{i=1}^N M_i$  test cases.

**Weak Diagonal IN Boundary Value Testing**

Weak diagonal IN boundary value testing complies with multiple fault assumption, too. But it tests cases when all parameters assume the same type and meaning of special values – if it is boundary values, all parameters assume minimal values or all assume maximal values, the same goes for inner OFF points or nominal point as it is shown in Fig. 9 (b) [41].

If the program has two parameters, test cases are generated according to the following algorithm.

- 1) Let us take boundary values of the first equivalence class of both parameters. We obtain 2 test cases (min, min), (max, max).
- 2) Let us consider inner OFF points of the first equivalence class of both parameters. We again obtain 2 test cases (min+, min+) and (max-, max-).
- 3) Nominal points of the first equivalence class of both parameters will give one test case (nom, nom).

Now we have 5 test cases for the first element in the set of Cartesian product of equivalence classes. For all elements, there will be  $5M_1M_2$  test cases.

- 4) Now exclude redundant test cases raised by overlapping boundary values and obtain  $5M_1M_2 - L_1L_2$  test cases.

There will be no less than  $5\prod_{i=1}^N M_i - \prod_{i=1}^N L_i$  test cases in the test suite generated by the weak diagonal IN boundary testing method for the program with N parameters.

If we skip step 4, we obtain that the weak diagonal IN boundary value testing method will give no more than  $5\prod_{i=1}^N M_i$  test cases.

**Multidimensional Boundary Value Testing**

The multidimensional boundary value testing method requires at least one test case for each boundary value of each equivalence class (Fig. 10 (a)) [1, 2, 5, 28–30, 32, 33].

In the case of N parameters, there will be  $\max_{i=1}^N (2M_i - L_i)$  test cases or no more than  $\max_{i=1}^N (2M_i)$  test cases if we do not have overlapping boundaries.

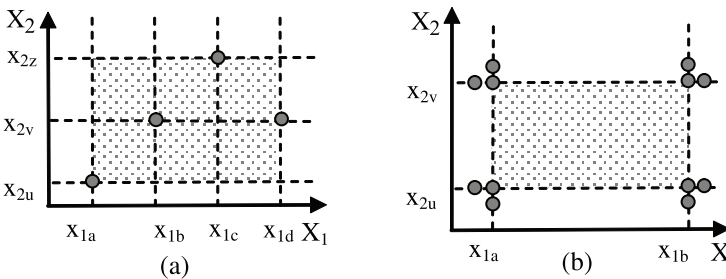


Fig. 10. Test cases for (a) multidimensional boundary value testing and (b) robust corner OUT boundary value testing

### Robust Corner OUT Boundary Value Testing

The robust corner OUT boundary value testing method tests cases when parameters assume boundary values or outer OFF points as it is shown in Fig. 10 (b) [42].

Let us count the test cases where one of inputs is OFF point.

If the program has only one parameter  $X_1$ , 2 test cases are obtained for each equivalence class of valid values – min- and max+. Because the count of equivalence classes is  $M_1$ , the count of test cases in the test suite is  $2M_1$ .

If the program has two parameters, test cases are generated according to the following algorithm.

- 1) We obtain  $2M_1$  test cases on each boundary of the second parameter, obtaining  $2M_1(2M_2 - L_2)$  test cases.
- 2) Repeat step 1 for each boundary class of  $X_1$  and obtain  $2M_2(2M_1 - L_1)$ .

Generalize this to  $N$  parameters –  $\sum_{i=1}^N (2M_i \prod_{\substack{j=1 \\ j \neq i}}^N (2M_j - L_j))$ .

The count of the test cases where all inputs are boundary values is  $\prod_{i=1}^N 2M_i$ .

So there will be at least  $\prod_{i=1}^N 2M_i + \sum_{i=1}^N (2M_i \prod_{\substack{j=1 \\ j \neq i}}^N (2M_j - L_j))$  test cases in the test suite

generated by the robust corner OUT boundary testing method for the program with  $N$  parameters.

The upper bound of the method's complexity is

$$\prod_{i=1}^N 2M_i + \sum_{i=1}^N (2M_i \prod_{\substack{j=1 \\ j \neq i}}^N 2M_j) = 2^N \prod_{i=1}^N M_i + \sum_{i=1}^N (2^N \prod_{j=1}^N M_j) = 2^N (N + 1) \prod_{i=1}^N M_i .$$

### Robust Strong Boundary Value Testing

The robust strong boundary value testing method [1] takes all boundary values, inner OFF values, and outer OFF values of equivalence classes of valid values and divides them into two sets – all valid values in one set and invalid values in the other set (Fig. 11).

The method allows to combine freely all values from the set of valid values in test cases.

The set of invalid values is revised. If there are two or more values from the same equivalence class, only one of them is left in the set. The method requires exactly one test case for each value that is left in the set where all the other values in the test case are valid values.

For each parameter, we have  $L_i$  overlapping borders. They will give  $3L_i$  valid values. We also have  $2M_i - 2L_i$  non-overlapping borders which will give exactly  $2M_i - 2L_i$  invalid values – outer OFF points that fall into classes of invalid values, and exactly  $2M_i - 2L_i$  valid values – inner OFF points of equivalence classes of valid values.

Hence, according to the method's adequacy criterion, valid values for  $N$  parameters will give  $\max_{i=1}^N (3L_i + 2M_i - 2L_i) = \max_{i=1}^N (2M_i + L_i)$  test cases, but invalid values  $\sum_{i=1}^N (2M_i - 2L_i)$  test cases.

The only question is about exactly  $2M_i - 2L_i$  boundary values. They can be valid if each interval of valid equivalence classes is closed or otherwise invalid.

Thus, the lower bound of the method can be achieved when all questionable values

$$\text{are valid: } \max_{i=1}^N (2M_i + L_i) + \sum_{i=1}^N (2M_i - 2L_i) + \max_{i=1}^N (2M_i - 2L_i).$$

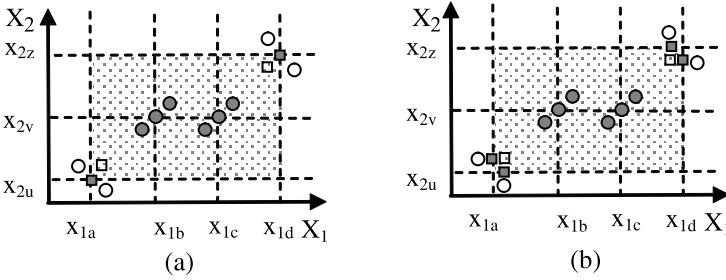


Fig. 11. Test cases for robust strong boundary value testing; (a) shows lower bound case, figure (b) – upper bound case. Filled dots represent test cases of overlapping borders, unfilled squares – test cases of inner OFF points of non-overlapping borders. Unfilled dots represent test cases of outer OFF points of non-overlapping borders. Filled squares show test cases of boundary values of non-overlapping borders when (a) they all are valid values, (b) they all are invalid values.

The upper bound of method can be achieved when all questionable values are invalid:

$$\max_{i=1}^N (2M_i + L_i) + \sum_{i=1}^N (2M_i - 2L_i) + \sum_{i=1}^N (2M_i - 2L_i) = \max_{i=1}^N (2M_i + L_i) + 4 \sum_{i=1}^N (M_i - L_i).$$

## 6 The Complexity and Subsumption Hierarchy of Domain Testing Methods

The majority of comparisons of testing adequacy criteria in related works use the subsume ordering. It was used to compare data flow testing adequacy criteria and several other structural coverage criteria [43–48]. Other ways of comparing the testing criteria have been studied and compared in [25, 49].

One of the formulations of subsume definition is the following: “let C1 and C2 be two software data adequacy criteria. C1 is said to subsume C2 if for all programs p under test, all specifications s and all test sets t, t is adequate according to C1 for testing p with respect to s implies that t is adequate according to C2 for testing p with respect to s” [49]. In other words, C1 subsumes C2 if every test suite generated by C1 is adequate for C2, too.

The hierarchy of domain testing methods according to subsume relation is showed in Fig. 12. Principles of test case generation for the corresponding criteria of each box are schematically showed in the figure.

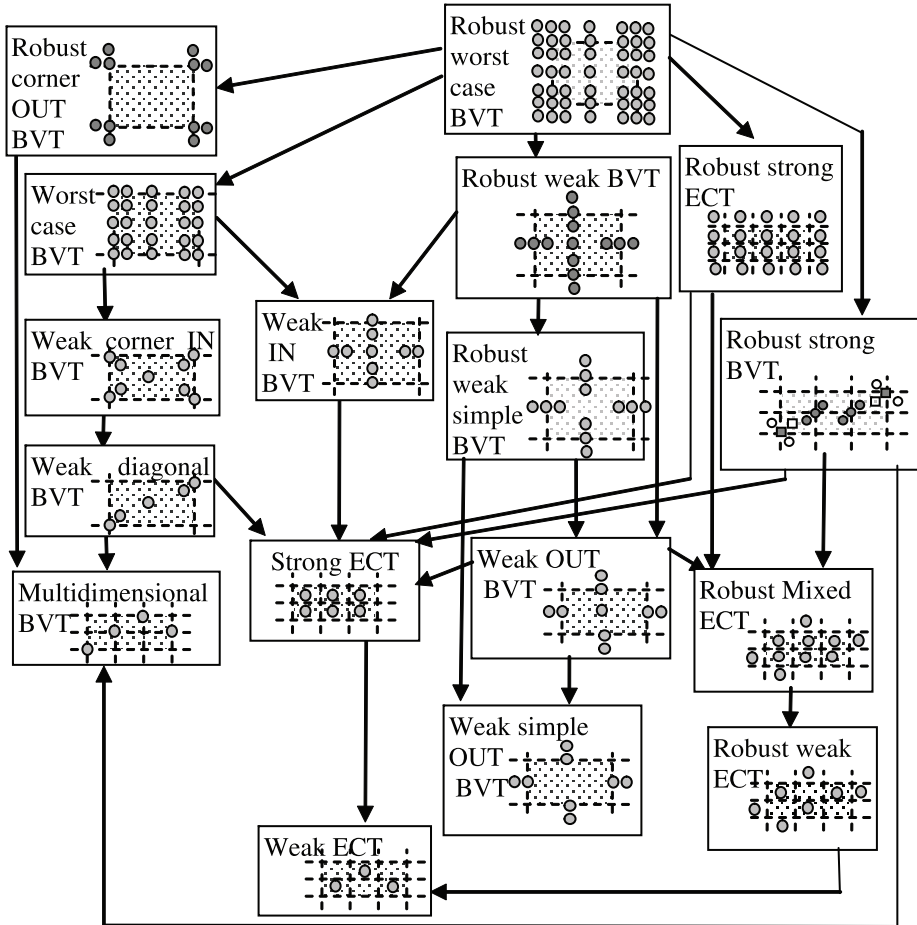


Fig. 12. The hierarchy of domain testing methods according to subsume relation

In most cases the subsumption is immediate and because of the limited space of the paper will not be discussed. Let us look at some cases.

1) Robust worst case BVT > Robust strong ECT

Robust strong ECT criterion requires a test case from each Cartesian product of each parameter's equivalence classes of valid values and equivalence classes of invalid values. Robust worst case BVT provides inner OFF points and nominal point from each Cartesian product of equivalence classes of valid values (points on boundaries are not suitable because there is a possibility that a boundary does not belong to equivalence class) and outer OFF points from Cartesian product in which equivalence classes of invalid values are involved. In the 2-dimensional case, it looks as showed in Fig. 13. Crosses and dots together are the test suite of robust worst case BVT criteria, but crosses alone are an adequate test suite of robust strong ECT criteria. In such a way, from each test suite of robust worst case BVT a test suite of robust strong ECT can be obtained.

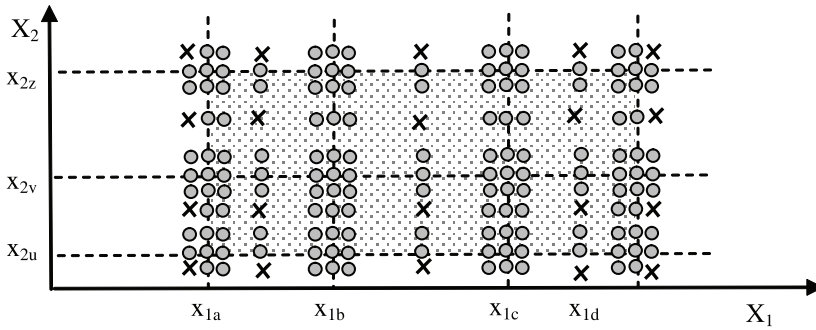


Fig. 13. Crosses and dots together are the test suite of robust worst case BVT criteria, but crosses alone are an adequate test suite of robust strong ECT criteria

2) Weak OUT BVT > Robust mixed ECT

Robust mixed ECT criterion asks for a test case from each element of Cartesian product of equivalence classes of valid values and from each class of invalid values. Weak OUT BVT provides nominal point for each element of Cartesian product of equivalence classes of valid values and outer OFF points for adjacent equivalence classes of invalid values. In the 2-dimensional case, it looks as showed in Fig. 14. Crosses and dots together are the test suite of weak OUT BVT criteria, but crosses alone are one of the adequate test suites of robust mixed ECT criteria. In such a way, from each test suite of weak OUT BVT a test suite of robust mixed ECT can be obtained.

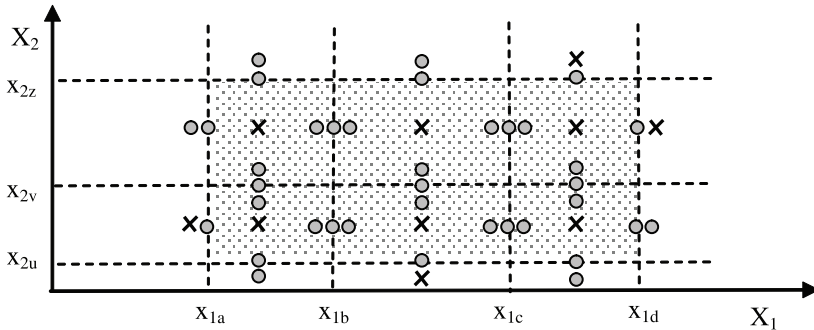


Fig. 14. Crosses and dots together are the test suite of weak OUT BVT criteria, but crosses alone are an adequate test suite of robust mixed ECT criteria

3) Weak simple OUT BVT does not subsume robust weak ECT.

Consider the situation in Fig. 15. Parameter  $X_1$  has one equivalence class for valid values with boundaries that do not belong to it ( $x_{1a}, x_{1b}$ ) and parameter  $X_2$  has the same situation – equivalence class ( $x_{2u}, x_{2v}$ ). Weak simple OUT BVT cannot provide a test case when both parameters assume valid values which is required by robust weak ECT.



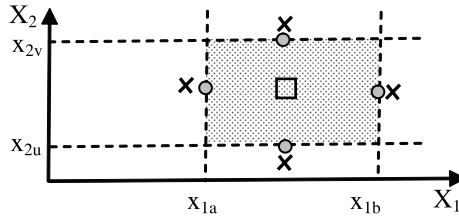


Fig. 15. Crosses and dots together are the testsuite of weak simple OUT BVT criteria. Crosses alone are candidates for the test suite of robust weak ECT criteria, while square represents the test case required by robust weak ECT that weak simple OUT BVT criteria cannot provide.

Table summarizes the lower and upper bounds of complexity of domain testing methods. It is easy to see that the robust worst case BVT method which is at the top of subsumption hierarchy in Fig. 12 also has the highest assessments of complexity. The methods in the lowest level of hierarchy have the lowest assessments of complexity.

It is obvious that if an adequacy criterion of method C1 subsumes an adequacy criterion of method C2, complexity of C1 is higher than of C2 of domain testing methods and their data adequacy criteria.

However, we cannot conclude that if method’s C1 complexity is higher than method’s C2 complexity, C1 subsumes C2. For instance, the complexity of the weak simple OUT BVT method is higher than the complexity of the weak ECT method. But the weak simple OUT BVT method tests a completely other kind of points than the weak ECT method; hence, the weak simple OUT BVT method does not subsume the weak ECT method.

Table

**The summarizing table of complexity of domain testing methods, where N – count of program’s parameters,  $M_i$  – size of the set of equivalence classes of valid values for parameter  $X_i$ ,  $Q_i$  – size of the set of equivalence classes of invalid values for parameter  $X_i$ , and  $L_i$  – size of the set of all common boundary values for parameter  $X_i$**

No.	Method	Complexity	
		Lower bound	Upper bound
1	Weak equivalence class testing	$N \max_{i=1} (M_i)$	The same as lower bound
2	Strong equivalence class testing	$\prod_{i=1}^N M_i$	The same as lower bound
3	Robust weak equivalence class testing	$\max_{i=1}^N (M_i) + \sum_{i=1}^N Q_i$	The same as lower bound
4	Robust strong equivalence class testing	$\prod_{i=1}^N (M_i + Q_i)$	The same as lower bound
5	Robust mixed equivalence class Testing	$\prod_{i=1}^N M_i + \sum_{i=1}^N Q_i$	The same as lower bound

6	Weak IN boundary value testing	$(4N+1)\prod_{i=1}^N M_i - \sum_{i=1}^N (L_i \prod_{\substack{j=1 \\ j \neq i}}^N M_j)$	$(4N+1)\prod_{i=1}^N M_i$
7	Weak OUT boundary value testing	$(4N+1)\prod_{i=1}^N M_i - \sum_{i=1}^N (L_i \prod_{\substack{j=1 \\ j \neq i}}^N M_j)$	$(4N+1)\prod_{i=1}^N M_i$
8	Weak simple OUT boundary value testing	$4N\prod_{i=1}^N M_i - \sum_{i=1}^N (L_i \prod_{\substack{j=1 \\ j \neq i}}^N M_j)$	$4N\prod_{i=1}^N M_i$
9	Robust weak boundary value testing	$(6N+1)\prod_{i=1}^N M_i - 2\sum_{i=1}^N (L_i \prod_{\substack{j=1 \\ j \neq i}}^N M_j)$	$(6N+1)\prod_{i=1}^N M_i$
10	Robust weak simple boundary value testing	$6N\prod_{i=1}^N M_i - 2\sum_{i=1}^N (L_i \prod_{\substack{j=1 \\ j \neq i}}^N M_j)$	$6N\prod_{i=1}^N M_i$
11	Worst case boundary value testing	$\prod_{i=1}^N (5M_i - L_i)$	$5^N \prod_{i=1}^N M_i$
12	Robust worst case boundary value testing	$\prod_{i=1}^N (7M_i - 2L_i)$	$7^N \prod_{i=1}^N M_i$
13	Weak corner IN boundary value testing	$(2^{N+1} + 1)\prod_{i=1}^N M_i - 2\sum_{i=1}^N (L_i \prod_{\substack{j=1 \\ j \neq i}}^N M_j)$	$(2^{N+1} + 1)\prod_{i=1}^N M_i$
14	Weak diagonal IN boundary value testing	$5\prod_{i=1}^N M_i - \prod_{i=1}^N L_i$	$5\prod_{i=1}^N M_i$
15	Multidimensional boundary value testing	$\max_{i=1}^N (2M_i - L_i)$	$2 \max_{i=1}^N (M_i)$
16	Robust corner OUT boundary value testing	$\prod_{i=1}^N 2M_i + \sum_{i=1}^N (2M_i \prod_{\substack{j=1 \\ j \neq i}}^N (2M_j - L_j))$	$2^N (N+1) \prod_{i=1}^N M_i$
17	Robust strong boundary value testing	$\max_{i=1}^N (2M_i + L_i) + \sum_{i=1}^N (2M_i - 2L_i) + \max_{i=1}^N (2M_i - 2L_i)$	$2 \max_{i=1}^N (M_i + L_i) + 2\sum_{i=1}^N (2M_i - L_i)$

## 7 Conclusions and Future Directions

Equivalence partitioning and boundary value analysis techniques are frequently mentioned in the books about software testing. In most cases, they describe common principles how to derive equivalence classes, boundary values, and present some

strategies how to make test cases from them. Each author or authors often emphasize one or two of the strategies but do not analyze their weaknesses or strengths, just demonstrate some examples. The positive exception is Jorgensen in [5].

This survey is an attempt to collect and describe some equivalence class testing and boundary value testing methods frequently mentioned in literature in a comprehensive manner. This paper assesses and summarizes complexity of domain testing methods and provides the hierarchy of domain testing methods according to subsume ordering. The complexity of testing methods is treated from the aspect of the size of the test suite generated by the method.

The author concludes, if some domain testing method subsumes other domain testing method, the first method also has higher complexity than the second method. However, the reverse statement is invalid.

This paper analyses the adequacy criteria of domain testing methods from three aspects – the kind of values to choose for testing, the data coverage principle, and the strategy how the chosen values are combined in test cases according to the data coverage principle.

Analysis of literature shows that methods with the smallest complexity are described in most cases. It shows that in praxis the most applicable are the methods that produce a not-too-big size of the test suite and are also practical – the methods that use single fault assumption in order to diagnose easier the cause of failure if it occurs.

Despite many sources of literature that cover boundary value analysis and equivalence partitioning, several important issues still remain largely unexplored. One of such issues is how the effectiveness and efficiency of testing is affected by the choice of data coverage criteria and the coverage of combinations of special or boundary values of different parameters of software.

The second issue related to the use of domain testing methods that is not adequately investigated is how to test the mutually dependent parameters of software better.

The third issue that needs further investigation is how the various models that describe the behavior of software can be used with the aim to refine equivalence classes of software input domain.

## References

1. Spillner A., Linz T., Schaefer H. (2007) *Software Testing Foundations. A Study Guide for the Certified Tester Exam*. Foundation Level, ISTQB compliant. Rocky Nook Inc.
2. Veenendaal E. (2002) *The Testing Practitioner*. UTN Publishers, Den Bosch, NL.
3. Beizer B. (1990) *Software Testing Techniques*. 2<sup>nd</sup> ed. New York: Van Nostrand Reinhold.
4. Beizer B. (1995) *Black Box Testing*. New York: John Wiley.
5. Jorgensen P. C. (1995) *Software Testing: A Craftman's Approach*. Boca Raton, London, New York, Washington D.C.: CRC Press.
6. Howden W. E. (1976) Reliability of the Path Analysis Testing Strategy. *IEEE Trans. Softw. Eng.* vol. 2, no. 3, 208–215.
7. Barzdins J., Bicevskis J., Kalnins A. (1975) Construction of complete sample system for correctness testing. In: Proc. MFCS 1975. LNCS, vol. 32, Berlin / Heidelberg: Springer, pp. 1–12.
8. Howden, W. E. (1975) Methodology for the Generation of Program Test Data. *IEEE Transactions on Computers*, vol. 24, no. 5, 554–560.
9. Clarke L. A. (1976) A System to Generate Test Data and Symbolically Execute Programs. *IEEE Trans. on Softw. Eng.* vol. 2, no. 3, 215–222.

10. Bičevskis J., Borzovs J., Straujums U., Zariņš A., Miller E. F. (1979) SMOTL – A System to Construct Samples for Data Processing Program Debugging. *IEEE Trans. Softw. Eng.* 5, 1, 60–66.
11. Richardson D. J., Clarke L. A. (1981) A partition analysis method to increase program reliability. In: Proceedings of the 5th international Conference on Software Engineering, IEEE Press, Piscataway, NJ, 244–253.
12. Richardson D. J., Clarke L. A. (1985) Partition analysis: a method combining testing and verification. *IEEE Trans. Softw. Eng.* SE-11, 12, 1477–1490.
13. Jeng B., Weyuker E. (1989) Some observations on partition testing. In: Proceedings of the ACM SIGSOFT '89 Third Symposium on Software Testing, Analysis, and Verification, ACM, New York, NY, 38–47.
14. Hamlet D., Taylor R. (1990) Partition Testing Does Not Inspire Confidence (Program Testing). *IEEE Trans. Softw. Eng.* 16, 12, 1402–1411.
15. Weyuker E. J., Jeng B. (1991) Analyzing Partition Testing Strategies. *IEEE Trans. Softw. Eng.* 17, 7, 703–711.
17. De Millo R., McCracken W. M., Martin R. J., Passafiume J. (1987) *Software Testing and Evaluation*. Benjamin-Cummings Publishing Co., Inc.
18. White L. J., Cohen E. I. (1980) A Domain Strategy for Computer Program Testing. *IEEE Trans. Softw. Eng.* 6, 3, 247–257.
19. Clarke L. A., Hassell J., Richardson D. J. (1982) A Close Look at Domain Testing. *IEEE Trans. Softw. Eng.* 8, 4, 380–390.
20. Afifi F. H., White L. J., and Zeil S. J. (1992) Testing for linear errors in nonlinear computer programs. In: Proceedings of the 14th international Conference on Software Engineering ICSE '92. ACM, New York, 81–91.
21. Jeng B., Weyuker E. J. (1994) A simplified domain-testing strategy. *ACM Trans. Softw. Eng. Methodol.* 3, 3, 254–270.
22. Zeil S. J., Afifi F. H., (1992) White L. J. Detection of linear errors via domain testing. *ACM Trans. Softw. Eng. Methodol.* 1, 4, 422–451.
23. Hajnal Á., Forgács I. (1998) An applicable test data generation algorithm for domain errors. In: Tracz W. (ed.) Proceedings of the 1998 ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA '98. ACM, New York, NY, 63–72.
24. Chien Y., Liu S. (2004) An Approach to Detecting Domain Errors Using Formal Specification-Based Testing. In: Proceedings of the 11th Asia-Pacific Software Engineering Conference, APSEC. IEEE Computer Society, Washington, DC, 276–283.
25. Weyuker E. J., Weiss S. N., Hamlet D. (1991) Comparison of program testing strategies. In: Proceedings of the Symposium on Testing, Analysis, and Verification, TAV4. ACM, New York, NY, 1–10.
26. Zhu H., Hall P. A., May J. H. (1997) Software unit test coverage and adequacy. *ACM Comput. Surv.* 29, 4, 366–427.
27. Grindal M., Offutt J., Andler S. F. (2005) Combination testing strategies: a survey. *Software Testing, Verification and Reliability*, 15, 3, 167–199.
28. Kaner C., Bach J., Pettichord B. (2002) *Lessons Learned in Software Testing: A Context Driven Approach*. New York, Chichester, Weinheim, Brisbane, Singapore, Toronto: John Wiley & Sons, Inc.
29. Culbertson R., Brown C., Cobb G. (2001) *Rapid Testing*. Prentice Hall PTR.
30. Kaner C., Falk J., Nguyen H. Q. (1999) *Testing Computer Software*. 2nd ed. New York, Chichester, Weinheim, Brisbane, Singapore, Toronto: John Wiley & Sons, Inc.
31. Myers G. J. (2004) *The Art of Software Testing*. 2nd ed., Hoboken, New Jersey: John Wiley & Sons, Inc.
32. Nguyen H. Q., Johnson B., Hackett M. (2001) *Testing Applications on the Web: Test Planning for Mobile and Internet-Based Systems*. 2nd ed., John Wiley & Sons, Inc.
33. McGregor J. D., Sykes D. A. (2001) *A Practical Guide to Testing Object-Oriented Software*. Addison-Wesley Longman Publishing Co., Inc.
34. Broekman B., Notenboom E. (2003) *Testing Embedded Software*. Addison-Wesley.
35. Link J., Frolich P. (2003) *Unit Testing in Java: how Tests Drive the Code*. Morgan Kaufmann Publishers Inc.
36. Watkins J. (2001) *Testing IT: an Off-the-Shelf Software Testing Process*. Cambridge, United Kingdom: Cambridge University Press.
37. Perry W. E. (2006) *Effective Methods for Software Testing*. 3rd ed. Hungry Minds Inc.
38. Craig R. D., Jaskiel S. P. (2002) *Systematic Software Testing*. Boston / London: Artech House Publishers.

39. Dustin E. (2002) *Effective Software Testing: 50 Ways to Improve Your Software Testing*. Addison-Wesley Longman Publishing Co., Inc.
40. Everett G. D., McLeod R., Jr. (2007) *Software Testing: Testing Across the Entire Software Development Life Cycle*. Wiley-IEEE Computer Society Press.
41. Gao J. Z., Tsao J., Wu Y., Jacob T. H. (2003) *Testing and Quality Assurance for Component-Based Software*. Artech House, Inc.
42. Copeland L. (2003) *A Practitioner's Guide to Software Test Design*. Artech House, Inc.
43. Rapps S., Weyuker E. J. (1985) Selecting software test data using data flow information. *IEEE Trans. Softw. Eng. SE*, 11, 4, 367–375.
44. Clarke L. A., Podgurski A., Richardson D., Zeil S. (1985) A comparison of data flow path selection criteria. In: Proc. 8th ICSE, 244–251.
45. Frankl P. G., Weyuker J. E. (1988) An applicable family of data flow testing criteria. *IEEE Trans. Softw. Eng. SE*, 14, 10, 1483–1498.
46. Ntafos S. C. (1988) A comparison of some structural testing strategies. *IEEE Trans. Softw. Eng. SE*, 14, 868–874.
47. Weiser M. D., Gannon J. D., McMullin P. R. (1985) Comparison of structured test coverage metrics. *IEEE Software*, 2, 2, 80–85.
48. Weiss S. N. (1989) Comparing test data adequacy criteria. *SIGSOFT Softw. Eng. Notes* 14, 6, 42–49.
49. Hamlet R. (1989) Theoretical comparison of testing methods. In: Proceedings of the ACM SIGSOFT '89 Third Symposium on Software Testing, Analysis, and Verification, TAV3. ACM, New York, 28–37.





LATVIJAS UNIVERSITĀTES RAKSTI  
751. sējums, DATORZINĀTNE UN INFORMĀCIJAS TEHNOLOĢIJAS

---

Latvijas Universitātes Akadēmiskais apgāds  
Baznīcas ielā 5, Rīgā, LV-1010  
Tālr. 67034535