Czech Technical University in Prague, FEL, Dept. of Computer Science
Charles University in Prague, MFF, Department of Software Engineering
VŠB–TU Ostrava, FEECS, Department of Computer Science
Czech Society for Cybernetics and Informatics
Czech ACM Chapter

Proceedings of the Dateso 2017 Workshop

Databases, Texts
DATESO
Specifications, and Objects

2017

http://www.cs.vsb.cz/dateso/2017/

acm Chapter Event

acm Chapter

ČSKI

avast foundation

April 10 – 12, 2016
Rančířov u Jihlavy, Czech Republic

DATESO 2017
© K. Richta, P. Moravec, editors

Technical editor:
Pavel Moravec, pavel.moravec@vsb.cz
VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Page count:	70
Contact:	Karel Richta
Tel. :	224357319, 604751846
Edition:	1st
First published:	2017
ISBN	978-80-01-06138-1
Electronic proceedings:	http://www.cs.vsb.cz/dateso/2017/

This proceedings was typeset by PDFLaTeX.

Published by Czech Technical University in Prague
FEL, Department of Software Engineering
Karlovo nám. 13, 121 35 Pague 2, Czech Republic

Steering Committee

Jaroslav Pokorný	Charles University, Prague
Karel Richta	Czech Technical University, Prague
Václav Snášel	VŠB-Technical University of Ostrava, Ostrava

Program Committee

Karel Richta (chair)	Czech Technical University, Prague
Mária Bieliková	Slovak University of Technology, Bratislava
Přemysl Brada	University of West Bohemia, Plzeň
Alena Buchalcevová	University of Economics, Prague
Miroslav Bureš	Czech Technical University, Prague
Karel Čemus	Czech Technical University, Prague
Jiří Dvorský	VSB-Technical University of Ostrava, Ostrava
Jan Genči	Technical University of Košice, Kosice
Irena Holubová	Charles University, Prague
Karel Ježek	University of West Bohemia, Pilsen
Radek Kočí	Brno University of Technology, Brno
Michal Krátký	VSB-Technical University of Ostrava, Ostrava
Marek Musil	College of Polytechnics Jihlava, Jihlava
Martin Nečaský	Charles University, Prague
Robert Pergl	Czech Technical University, Prague
Jaroslav Pokorný	Charles University, Prague
Václav Řepa	University of Economics, Prague
Václav Snášel	VSB-Technical University of Ostrava, Ostrava
Jiří Sochor	Masaryk University, Brno
Jan Staudek	Masaryk University, Brno
Pavel Strnad	Czech Technical University, Prague
Petr Šaloun	VSB-Technical University of Ostrava, Ostrava
Michal Valenta	Czech Technical University, Prague
Peter Vojtáš	Charles University, Prague

Organizing Committee

Karel Richta	Czech Technical University, Czech ACM Chapter, Prague
Božena Mannová	Czech Technical University, Czech ACM Chapter, Prague
Pavel Moravec	VŠB-Technical University of Ostrava, Ostrava

Preface

DATESO 2017 is the international workshop on current trends on Databases, Information Retrieval, Algebraic Specification and Object Oriented Programming and was held on April 10 – 12, 2016 in Rančířov u Jihlavy, Czech Republic.

The 17$^{\text{th}}$ year was organized by Department of Computer Science, FEL ČVUT Praha with the cooperation of Department of Software Engineering MFF UK Praha, Department of Computer Science VŠB-Technical University Ostrava, Working group on Computer Science and Society of Czech Society for Cybernetics and Informatics, and Czech ACM Chapter.

The DATESO workshops aim for strengthening connections between these various areas of informatics.

The proceedings of DATESO 2017 are also available at DATESO Web site: `http://www.cs.vsb.cz/dateso/2017/`. The Program Committee selected 6 papers from 9 submissions, based on three independent reviews.

We wish to express our sincere thanks to all the authors who submitted papers, the members of the Program Committee, who reviewed them on the basis of originality, technical quality, and presentation. We are also thankful to the Organizing Committee. Special thanks belong to Czech Society for Cybernetics and Informatics, and to Czech ACM Chapter.

Our thanks go also to Pavel Moravec who, as copy editor of DATESO Proceedings, helped to prepare this volume and provided technical support for the conference preparation portal.

April, 2017 K. Richta, J. Pokorný,V. Snášel (steering committee)

Table of Contents

Impact of User's Emotion on Software Adaptation[1]

Jiří Šebek and Karel Richta

Department of Computer Science and Engineering, Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo nám. 13, 121 35 Praha 2, Czech Republic
{sebekji1,richta}@fel.cvut.cz

Abstract. The Adaptive Application Structure (AAS) software design resolves the problem with different users needs. It changes its structure based on the current context. Every user has different needs so it is quite challenging problem because the structure of application can be created with quite large number of options. The AAS-approach advantages are clear in form of flexible software design. Developer does not have to create different types of application based on users needs. This approach removes the problem of user modeling (e.g. Personas) for UI developers. As a resource of information about user we can use not only filled information or information about his behavior. We can use also context that will focus on his emotions while he uses the application. This paper considers emotions as a valid and important part of context-aware design.

Keywords adaptive application structure, context-aware design, reduced maintenance and development efforts, user´s emotion

1 Introduction

In the last century mankind has entered a new stage of development of science, where manual work began to be replaced by intelligent mechanisms. Machines are becoming more self-sufficient because these machines are more equiped with electronics these days. Since then all devices only stronger links with our lives and now they aim not only to help man at work, but also provide convenience, comfort and facilitate daily routine. At the same time communications equipment becomes a part of everyday life. It is hard to imagine ourselves without helpers such as computer and mobile phone. Now it is standard attributes of life such as clothing and shoes. A large percentage of mobile usage falls under smartphones. When such tools that sim-

1 This work has been partially supported by the Grant Agency of CTU No. SGS15/210/OHK3/3T/13 and partially also by the AVAST Foundation

plify life are invented, we can improve them to be easier to use, attractive, more specified to user needs. We can also aim to improve not only the device itself, but also the applications that run on it. Nowadays there are a lots of applications. User expects not only a working application, the actual efficiency is no longer enough. User wants more in form of a nicely decorated product that is able to understand its owner and adapt to user´s needs (visually and functionally).

This survey does not seek to cover every technique and research project in the area of software adaptation, context-aware design. Instead, it hopes to serve as an introduction to this rapidly evolving field, bringing up interesting ideas.

This paper is organized as follows. Section 2 describes the background and terminology of this paper. Related work is included into Section 3. Sections 4 contains analysis of emotions as source of information, goals, general information, measurement, error detection, consistency of color binded with emotions. Section 5 describes structure adaptation and introduces design of new framework. Section 6 presents conclusion and future work.

2 Background

Software applications provide User Interface (UI) for the interaction with users. These days UI's are very important and their primary goal is to offer the best possible comfort service. There arises a problem with increasing number of different users with special requirements and abilities. The number of different electronic devices and the number of different environments with different types of users grows. It is necessary to create specific UIs that can adapt to the particular user or more generally a context. The context characterizes the situation where the application is used. It may include the current environment, device properties, by which user controls the application and the capabilities and preferences of the user. Dey defines context as [1]:

Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

Also there are another definitions. The definition that Dey provided [1] is the most general one. The first definition of context that were made by Schmidt [2] is:

Context awareness as knowledge about the user's and IT device's state, including surroundings, situation, and to a less extent, location.

The software structure is equally important as UI. Not only that user wants to see the concrete information for him but also he wants to find it fast. In general it is impossible to predict what user wants to do or gets from application and render that information in the first screen of application. One user can use the application as whole unit. So all functions are used by user. The other user wants to use only some of the application functions. Here the application structure should be more *user-friendly* for functions that user wants.

The applications that are using context information (e. g. adaptation) is called context-aware. In Figure 1 you can see the example of context-aware application. The AAS has similar problems as Adaptive User Interface (AUI). Both uses context to adapt structure or in AUI case UI part of application. For a context dependent application it is necessary to expend a lot of resources to the development and maintenance of the AAS or AUI descriptions. The main problems are the complexity, content volume and the number of interactive elements. An example of this problem is the construction of a multiple platform application. We often have to create several separate applications or subsystems, because there are different technologies or different interaction with a particular device. This results in increased costs, low flexibility to contextual situations and complex adaptation to changes of the underlying application. These changes must be done separately for each platform.

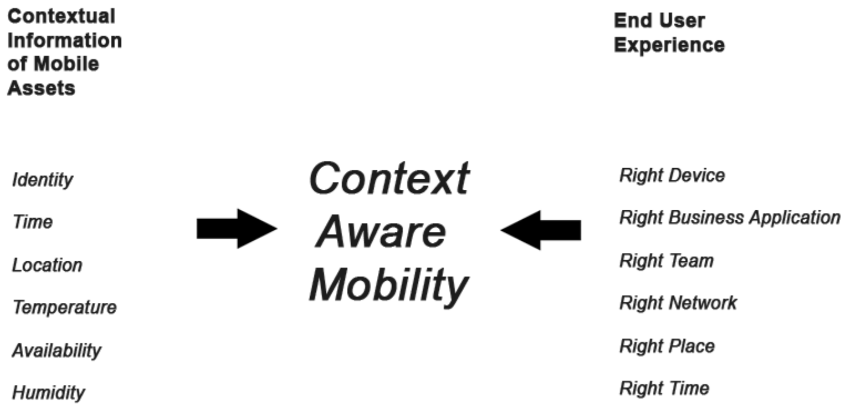


Fig 1. Context-aware mobility

As we defined the term context in the paragraph above, it can be divided into four parts [3]. Classification is divided into two categories according to the type of changes to *static* and *dynamic*. If the parameter, that application gets into context, is changed during the session (time that application is used), it is assigned a *dynamic* parameter, otherwise as *static*. The next level of classification is divided into *functional* and *presentation*. The *functional* category contains parameters that change functionality of backend. The second category contains parameters that change the frontend. Table 1 shows an example.

Table 1. Placement of context parameters into categories

	Static	Dynamic
Functional	Role, identity, platform, preferences	Connection to the internet, time, location
Presentation	Platform, preferences	Lighting, activity of user, orientation of device

3 Related work

Approach in the articles [4][5] is not the only way how to generate User Interfaces (UI) from the model. The topic of UI generated from domain objects is mentioned in [6][7]. The framework is called Meta-widget and it is based on Model-Driven Development (MDD). The user just creates objects and puts them to Meta-widget's framework. The UI is generated according to the model. Meta-widget supports a lot of technologies from Android, Google Web Toolkit (GWT), HTML 5 (POH5), JavaScript to JSF and JSP. Meta-widget works in three basic steps. First, Meta-widget comes with a UI component native to your existing frontend. Second, Meta-widget inspects, either statically or in the run-time, your existing back-end architecture. Third, Meta-widget creates native UI subcomponents matched to the back-end. In articles [6][7], the other aspects were added based on annotations. Meta-widget adds this information to UI based on existing back-end of any applications. Aspect-oriented User Interface Design for Android Applications MDD is based on the idea that the model should be primary centralized place for all information. This model is then compiled or transformed to a source code that is determined the application. The benefits are reduction of information in application and concentration of the structure of information into one place. The disadvantages can be adaptation and evolution management [8]. This approach does not deal well with OOP, because we need to maintain the interconnection between multiple models with the back-end of the application. There exist other approaches, how to describe information, such as through the Domain-Specific Languages (DSL). Sometimes, they are informally called mini-languages, because they describe the additional information inside the other language. There are a wide variety of DSL.

Generative Programming (GP) is a specific type of a programming that generates the source code from domain specific code. The goal is to improve productivity of development, put together the advantages from multiple approaches, such as integrating application code and domain model. Furthermore, it support reuse, adaptation, and simplify management of components [9].

Currently there is only a single framework for Android [10], which is aimed at creating adaptive style of user interface in AOP. Android framework [10] does not use the context of the sensors, so its integration brings multiple challenges.

Various implementations of self-adaptation exists [11][12][13]. They use different approaches.

The Rainbow framework [11] uses software architectures and a reusable infrastructure to support self-adaptation of software systems. The use of external adaptation mechanisms allows the explicit specification of adaptation strategies for multiple system concerns.

Another paper [12] suggests to use Petri nets for modeling the behaviors that change at run-time in response to environmental changes. It is an extension of hybrid Petri nets by embedding a neural network algorithm into them at some special transitions.

Another framework [13] is focused on Service-oriented enterprise systems. They propose a model-driven approach for the dynamic adaptation of Web services based on ontology-aware service templates. Service templates are based on OWL-S descriptions.

All of these approaches [11][12][13] are focused on back-end systems and none of them deal with UIs. UIs of applications are also very important and we cannot omit it. The front-end development is quite different from back-end and needs the re from users point of view.

LILOLE framework [14] is focusing on concept for lifelong learning from sensor data streams for predictive user modelling that is applicable in scenarios where simpler mechanisms that rely on pre-trained general models fall short. It gains information from senzor data. This framework is helping with aspects between user and device in some situations. It uses machine-learning proccess to learn from user. The evaluation application is focused on Instant Messaging and users status.

4 Emotions

According to Paul Ekman [15] There are seven basic emotions: Disgust, Anger, fear, Sadness, Happiness, Surprise, Contempt. We can use these emotions in order to adapt software. The applications can adapt their UI based on users emotion. UI can adapt smoothly and rarely, the likelihood of concern or failing color palette customer is a minor than if the UI was fixed or adapt with rapid speed. A design that matches the normal emotions do not causes disagreements with internal state man. Consciously person can ignore the present disharmony, but subconsciously perception of colors associated with pleasure is not pleasant when someone is sad, and vice versa. One of the most important factor is when negative emotions occur regularly and always in the same place. It will hardly be considered as a coincidence. Found results could provide signal to developers that are likely to be some problems in functionality of application. The advantage is that we immediately known that part of the application where these problems occur.

4.1 measurement of emotions

There are a number of APIs that lead with a camera to capture the emotions. One of them is Affectiva Emotion SDK It is supported on Android devices, moreover does not need a connection to the Internet (the analysis will take place directly on the device). Affective 7 can reveal emotions [4].

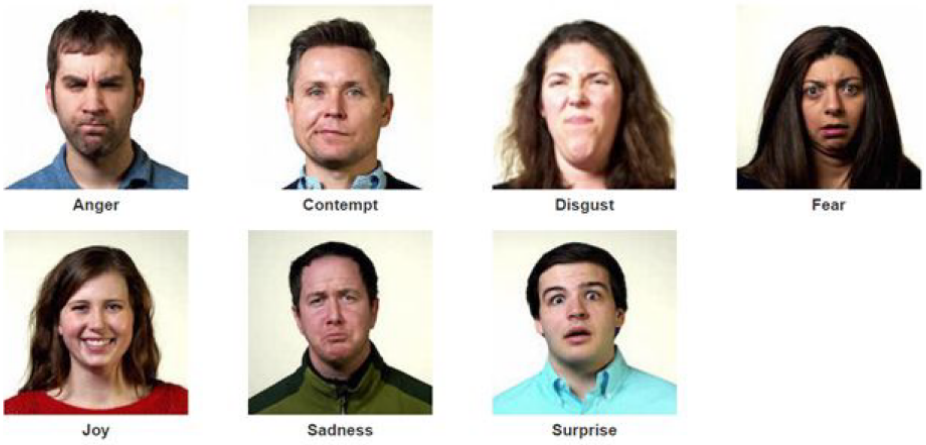


Fig 2. Affectiva emotions [4]

Metrics that uses the SDK show when a user demonstrates a certain emotion or expression faces (e.g., smile), along with the degree of expression of: from 0 to 100 according to how strongly the expressed concerned expression.

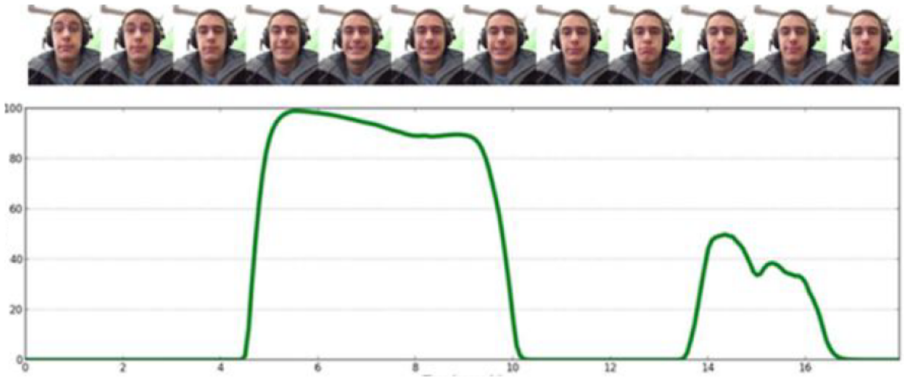


Fig 3. The output data from the labor Affective [4]

4.2 Error detection emotions

Emotions can not detect absolutely right. It is working with the human factor and, therefore, deviations are necessary. So maybe some people have specific facial expressions for any expression feeling, which according to the "template" shows often the opposite thing. For this reason we say emotion likelihood and impact of facial expression on this probability, which can be either positive or negative.

4.3 Consistency of color and emotion of man

Alexander Etkind (1979, 1980-1985) conducted series of studies colored emotional connected among adults. In the first trial, in 1979, it was examined the connection 8 colors of the test M. Luscher with 9 the fundamental emotions by K. Izard (1980). The following table shows the percentage of frequency color associations with emotional factors Izard [8].

Table 2. Percentage of frequency color associations with emotional factors [8]

Color	In-terest	happin ess	Surprise	Sadness	Anger	Disgust	Sha me	Fear	Tired-ness
Gray	6	4	2	27	1	15	18	12	53
Blue	27	4	2	27	5	7	13	15	8
Green	26	10	26	13	8	7	19	8	7
Red	16	52	23	4	55	4	4	17	2
Yellow	20	24	56	1	9	19	12	15	1
Purple	5	12	14	12	6	22	16	7	12
Brown	10	8	3	14	4	28	17	3	23
Black	10	2	2	22	38	18	13	43	24

5 Structure adaptation

In the most cases the application structure is static. In order to create adaptation for the structure it is natural to save it in some meta-model. For this approach we can use AOP. It will reduce our effort to develop such application. When we have set some application structure we save it in meta-model (in the most cases it is some type of cache). Now how to display it in the way user wants. Information grouping is term which specifies the process of information gathering into groups. The simple applications can have structure that can be represented by simple list. The more complex ones can be represented by:

- Menu
- Tab folders

- Collapsible blocks
- Different dialogs with navigation
- Wizard based UI
- Tree view

From the list above we can see that this list is primarily targeting desktop applications. In Android mobile application there is structure called Fragments instead of Tab folders. Collapsible blocks are very unacceptable in mobile platform. There is small space for layout, UI elements and it should not move to disturb or confuse the end-user. Different dialogs with navigation has same problem on mobile platform also. The contexts are different. In the web applications we can get some information from browser cache. Mobile devices can give us more information (e. g. from sensors, devices hardware). The most used tool to represent structure is menu. In the list below there is classification based on end-device:

- Desktop (Web) application
- Mobile application

The meta-model of application structure can be represented by graph. A graph is made by two sets called *Vertices* and *Edges*. It may look like tree but tree is undirected graph and as we can see in Figure 3 on the left side it is directed graph. We can see the similarity between menu structure and tree structure. In Figure 4 on the right side we can see the menu we will not considering as structure of menu. The main reason is fact, why should developer make a loop in menu like this. It is redundant link and user will just be confused why there is same item placed in two spots. We have to take a note the structure we are presenting in this paper is navigation structure. If there is a link inside the node in menu (program screen) it is alright but do not need to visualize this information in navigation structure.

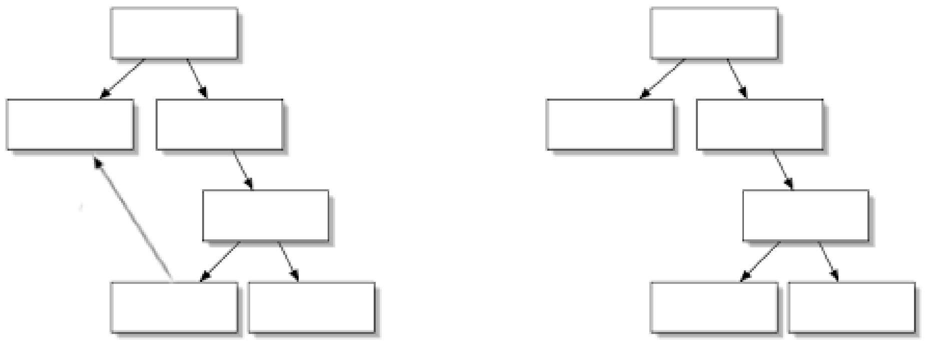


Fig 4. Graph representations of application structure

Vertices are represented by nodes. Nodes is divided into: *root*, *inner* and *leaf. Root* is only one per menu. *Inner* node can represent just group information or clickable link to page. *Leafs* are always links.

5.1 Mobile platform - Android

In the chapter above it was mentioned types of nodes. We can divide this into more detailed by platform classification:

- Desktop (Web) application
  - Inner node can represent just group information or clickable link to page.
- Mobile application
  - Inner node can represent just group information not clickable link to page.

Another limit comes up on Android platform. The depth of menus that can be created is only 3 (if we count root as well).

For dealing with adaptive structure we can use framework from [16]. We can extend his meta-model by structure that will save application structure as you can see in Figure 5. The new package in design is called structure in this *UML* diagram. This is just a part of *UML* diagram from [16].

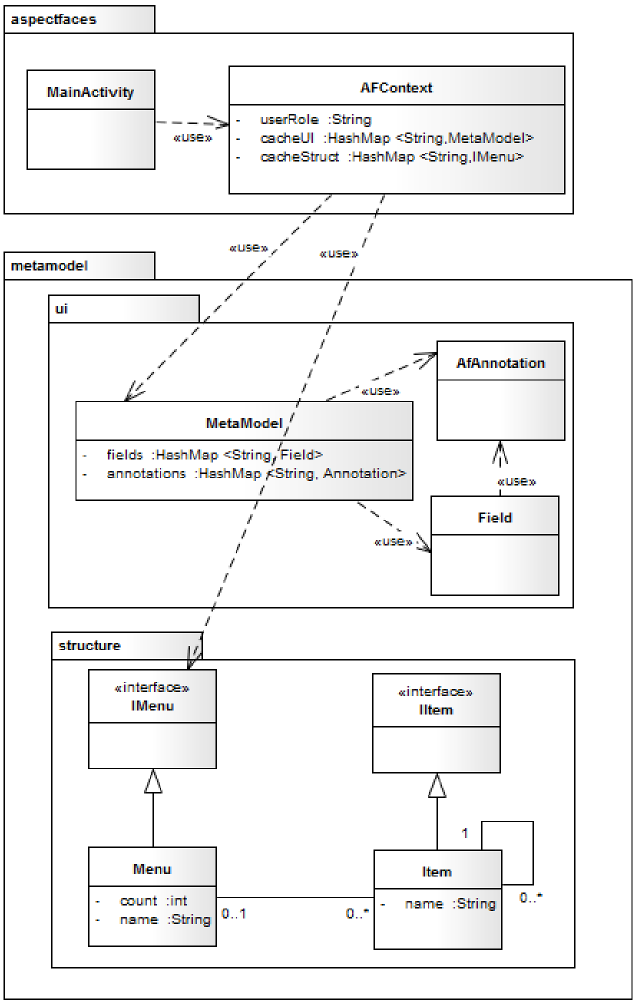


Fig 5. Meta-model used in extended framework

In this paragraph we can define two instances of this structure adaptation:

- User that uses only specific branch of nodes
- User that uses all nodes

User from the first category needs to reach the used nodes fast as possible. So if it is possible we need to move them up in our structure. Also there is subproblem. If there is a lots of leaf nodes in one place. This long list is hard to list and time consuming specially on mobile devies. The question is if solution should divide this leaf nodes from one parent into two parts and create another parent node for the second part. The problem will come up in naming the second parent. Also how should user recognize in which of these two parent nodes will he find his leaf node. User from the second category needs to reach all nodes in the same average time. So if it is possible we need to adjust structure to be as ballanced as possible. In figure 5 there are two ideal structures. The gray filled nodes are ones that user is using. On the left side of figure 5 there is structure for user that is using only one node. As we can see this node is as much on the top of the structure. The rest of structure can remain same. On the right side of figure 6 there is ideal structure for user that is using all nodes. This show just basic concept. If there will be more nodes it will represents list and it is not ideal. For more nodes it is required to let some part of original structure there and create more complex structure that will be balanced.

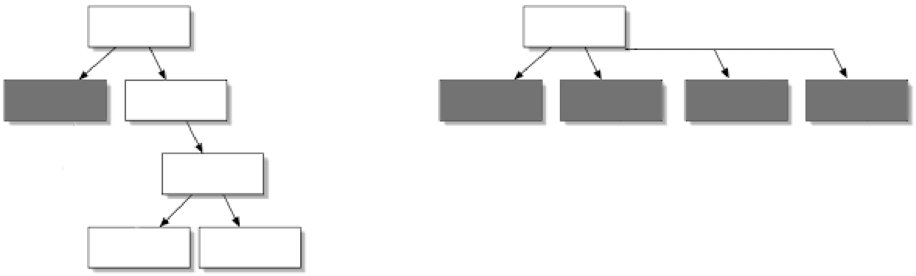


Fig 6. Ideal structure for specific type of users

The same priority as solving the problem itself has also how should work the algorithm in lifetime of application. If the structure will change every time user will start application it will confuse him and user will refuse to use it. It is needed to set number of application starts and then finds out how adjust application structure. Or it is needed to set period in which will framework gather information. The first choice seems like to be the better one, because we can be sure that user used application. In the second choice we do not know if user use it or how much. As a result algorithm will have two phases:

- Learning phase
- Production phase

This princip of two phases is used in distributed algorithm specially in *Leader election*. In second phase there will be also short period of users learning phase. User will have to adapt for the better structure because he will be adapted to the old structure.

Learning phase has some difficulties too. We do not know how find out if users behavior was exception and when it is users ordinary behavior pattern. The boundary is not known for now.

In figure 7 we can see model that is based on information above. First section is divivded by data we can get (filled by user, sensors, other devices, behavior, emotions). The second part represents how we represent these data and how we save them. In other words what does data mean. The last part is focused on which part of software we can adapt to help user in using the application.

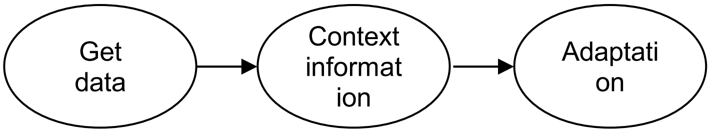


Fig 7. Adaptive model design

6 Conclusion and Future Work

In this paper, we have presented ways how to handle AAS and possible extension of AOP-based framework from [16]. There are lots of classifications of this problem. We demontrate the main ideas of this methodology. We classified emotions as a important information. Based on that we can adapt software to users needs. The future work will have two important steps. The first one will be to implement extension for desktop applications and then for mobile applications. The second importnat step will be to test it with real users. The expectation is that applications with AAS will be more *user-friendly* and users will have better feeling while using it.

References

1. G. Abowd, A. Dey, P. Brown, N. Davies, M. Smith, and P. Steggles, “Towards a better understanding of context and context-awareness,” in *Handheld and Ubiquitous Computing, ser. Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1999, vol. 1707, pp. 304–307.
2. Schmidt, A., Aidoo, ,K.A., Takaluoma, A., Tuomela, U.: Advanced Interaction in Context. In: *Handheld and Ubiquitous Computing, series Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1999, vol. 1707, pp. 89-101. ISBN 9783540481577.
3. Černý, T., Čemus, K., Donahoo, M. J., and Song, E.: Aspect-driven, Data-reflective and Context-aware User Interfaces Design. In: Applied Computing Review, Vol. 13, Issue 4, ACM, New York, NY, USA, 53-65. ISSN 559-6915,
4. Černý, T., Donahoo, M. J., and Song, E.: Towards effective adaptive user interfaces design. In Proceedings of the 2013 Research in Adaptive and Convergent Systems (RACS ? 13). ACM, New York, NY, USA, 373-380. DOI=10.1145/2513228.2513278, http://doi.acm.org/10.1145/2513228.2513278 (2013)
5. Černý, T., Čemus, K., Donahoo, M. J., and Song, E.: Aspect-driven, Data-reflective and Context-aware User Interfaces Design. In: Applied ComputingReview, Vol. 13, Issue 4, ACM, New York, NY, USA, 53-65. ISSN 559-6915, http://www.sigapp.org/acr/Issues/V13.4/ACR-13-4-2013.pdf (2013)
6. Kennard, R. and Leaney, J.: Towards general purpose architecture for UI generation. Journal of Systems and Software, 83(10) http: / / metawidget . sourceforge . net / media / downloads / Towards a General Purpose Architecture for UI Generation. Pdf (2010) 1896-1906

7. Kennard, R. and Robert, S.: Application of software mining to automatic user interface generation. In SoMeT?08. http: / / metawidget . sourceforge . net / media / downloads / Application of Software Mining to Automatic User Interface Generation.pdf (2008) 244 - 254
8. Morin, B., Barais, O., Jezequel, J.-M., Fleurey, F. and Solberg. A.: Models@run.time to support dynamic adaptation. Computer, 42(10) (Oct. 2009) 44-51
9. K. Czarnecki, "Overview of generative software development," in Unconventional Programming Paradigms, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, vol. 3566, pp. 326–341. [Online]. Available: urlfhttp://dx.doi.org/10.1007/ 11527800n 25g
10. Šebek, J.: Aspect-oriented user interface design for Android applications, diploma thesis. Department of Computer Science, CTU FEE, Prague (2014)
11. D. Garlan, S. W. Cheng, A. C. Huang, B. Schmerl and P. Steenkiste, "Rainbow: architecture-based self-adaptation with reusable infrastructure," in Computer, vol. 37, no. 10, pp. 46-54, Oct. 2004. doi: 10.1109/MC.2004.175, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1350726&isnumber=29695
12. Ding, Z.; Zhou, Y.; Zhou, M. "Modeling Self-Adaptive Software Systems With Learning Petri Nets", Systems, Man, and Cybernetics: Systems, IEEE Transactions on, On page(s): 483 - 498 Volume: 46, Issue: 4, April 2016
13. Staikopoulos, Athanasios; Cli#e, Owen; Popescu, Razvan; Padget, Julian; Clarke, Siobhan "Template-Based Adaptation of Semantic Web Services with Model-Driven Engineering", Services Computing, IEEE Transactions on, On page(s): 116 - 130 Volume: 3, Issue: 2, April-June 2010
14. Fetter, M. and Gross, T.: LiLoLe-A Framework for Lifelong Learning from Sensor Data Streams for Predictive User Modelling. In: Human-Centered Software Engineering: 5th IFIP WG 13.2 International Conference, HCSE 2014, Paderborn, Germany, September 16-18, 2014. Proceedings, p. 126-143. DOI=10.1007/978-3-662-44811-3 8, http://dx.doi.org/10.1007/978-3-662-44811-3 8 (2014)
15. HODAKOV, Viktor. Adaptive User Interface: задачи исследования и построения - Восточно-Европейский журнал передовых технологий №2, 2004. – С. 20-29.
16. Šebek, J. - Richta, K.: Aspect-oriented User Interface Design for Android Applications. In DATESO 2015. Prague: MATFYSPRESS, 2015, p. 121-130. ISSN 1613-0073. ISBN 9788073782856

Experiences with Data Lineage Metadata Storing in Relational and Graph Database

Karel Quast and Michal Valenta

Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9, 160 00 Prague 6, Czech Republic
{karel.quast, michal.valenta}@fit.cvut.cz

Abstract. We've spent few last years researching possibilities of storing and managing metadata for data lineage. We tried using relational database and special category of NoSQL databases - graph database. This paper describes our experiences with two competing products for data lineage and possibilities to store and refresh data lineage metadata. We present research in progress, including benchmarks and we're currently working on several challenges, that customers requested, i.e. adding temporal dimension and multiple hierarchical views, such as logical/physical.

Keywords: graph database, relational database, data lineage, metadata storing

1 Introduction

This paper comes out of our research, regarding storing metadata for database data lineage graphical representation. This operation is usually used in Data Warehouses, where's expected to have very large databases and raises need for graphical representation of database metadata.

Term "*data lineage*" [1] refers to "*Data Provenance*", which is referred as "*Record trail that accounts for the origin of a piece of data (in a database, document or repository) together with an explanation of how and why it got to the present place*". In large database with thousands of ETL ("*Extract Transform Load*") operations, it takes a lot of time to discover, how specific value at specific table is filled/calculated. Modern data lineage visualization tools can show exact path of data flow from source tables to target tables, which saves large amount of time examining all path related ETLs.

Formally, data lineage can be introduced [2] as input set of source data **I1,. . . ,Ik**, which enters transformation graph **T1,. . . ,Tn**, which results into output set of data **O1,. . . ,Om**. In this context, individual transformations are examined as data lineage.

There are several commercial products specialized for visualization of data lineage, some sold as specialized software (i.e. *Manta*[1], *SQLdep*[2], *E/R studio*),

[1] https://getmanta.com/
[2] https://sqldep.com/

others as part of bigger package (i.e. *Informatica*). Our paper focuses on first two mentioned, since we have established deeper connections with developers and are able to research ways to make storing metadata for data lineage more effective and more capable.

We're currently researching and benchmarking storing data lineage metadata in graph database, which seems natural, since data stored are natural graph. Unfortunately graph databases are very young tools and are still being developed, which brings many challenges to our work.
In this paper, we share some results of running research on this topic, i.e. Relational database x Graph database benchmarks, multiple views in one database, temporal dimension and we describe our focus on other issues/features.

The rest of this paper is organized as follows: comparison of two different approaches in two different products (Manta and SQLdep), details of implementation metadata repository in a graph database and detailed description of current and future research we're working on.

2 Basic characteristics of Manta and SQLdep software

Both tools, we had the chance to examine closer share the same goal – visualize data lineage. To reach this goal, both tools have to perform several steps:

1. acquire metadata from source database(s) consisting at least of tables, views, synonyms, descriptions, sql script and packages,
2. parse extracted metadata and "understand" data flows,
3. store parsed metadata into own database,
4. visualize data lineage on user request.

As addition, more than one database can be processed and visualization can cover data lineage from primary source of data through multiple physical databases to final data consumer.
Level of visualized details can be from top level (technology) to the bottom (column name) creating hierarchy "*technology – database – schema – table – column*". In certain cases, where target data are used in stored procedures as output rows, hierarchy would be "*technology – database – schema – package – procedure – parameter*".

Regardless of visualized hierarchy, the inner storage of data lineage software have to use universal structure to cover any possible output combination. Database design has to be able to quickly respond for user requests and provide all information for visualizer to complete final data lineage graph for user. At this point, Manta and SQLdep differs.

In the text bellow, we work with terms "*basic graph*", "*expanding path*" and "*base view*". When talking about basic graph, we mean graph build directly by current filters. User usually starts at some point in graph and use some restrictions for displaying data lineage. This starting point and filters are applied and basic graph (a subgraph of metadata database) is calculated. Afterwards, user can decide to follow some lineage/path outside boundaries of basic graph, which

we refer as expanding path. Base view is basic graph simplified for visualization purposes. Application calculates basic graph and shows only some parts of it to fit application screen. If user choose to see more details, application uses base graph and calculates another customized view of basic graph.
SQLdep uses relational database stored in cloud to have sufficient performance in case of compiling complicated graphs. On each request, basic graph is built from scratch, so expanding path through visualized graph requests same data from database enriched of expanding path (original filters are expanded). The main advantage of this approach is faster basic graph response, but slower path expansion, when user decides to follow some specific path.

Manta used relational database in earlier versions, nowadays graph database is being used and stores data as embedded database on customer's server. This evolution is tightly connected to our research work. Along with using different architecture on metadata storing, visualization approach is different too. On user request base view is built. From this base view, final visualization is calculated and simplified data lineage is shown to user. If user decides to expand some path in graph and this expansion is inside base view boundaries, calculations are done without requesting data from Manta database. Advantage of this approach is faster path expanding, but slower initial response, because base view has to be requested and calculated.

Both approaches are tight with database technology used, because building base view on relational database proved to be very slow and from user point of view even unacceptable. Implementing graph database as main Manta storage enabled using base view technology on larger databases within acceptable response time.

3 Implementing graph database

As we mentioned before, in earlier versions, Manta used classical relational database (specifically PostgreSQL), which was easy to implement and more proven method. On larger databases, building base view took several seconds, which became unacceptable. Using more performing hardware was not considered as a solution, mainly from two reasons: clients are not willing to invest into data lineage tools large sums of money and it would be very short-term solution, since databases are growing larger every day. This issue led us to start our research on using graph databases and proven to be promising.

Graph databases in general are very young and rapidly developing. At the beginning of our research, there wasn't even standardized API to access graph database. Our experiments started with Neo4J. We used anonymized client database consisting of ~ 1 million nodes and 3 million edges. Space required by Neo4J was ~ 25 % greater than original PostgreSQL, but with growing data amount was growing linear. Initial data import took $\sim$ same time (within 10 % of difference in behalf of PostgreSQL).

Main difference came with queries. As expected, Neo4J was much faster on complicated queries (i.e. queries returning longer paths) than PostgreSQL. Using of native Java API was needed, since Neo4J language Cypher generated significantly longer response times. More on this topic is described in chapter 4 of this paper.

Next wave of experiments used graph database Titan. Specialty of this solution lies in possibility of binary storage selection. Titan works with PersistIt, BerkeleyDB and Cassandra, after considerations PersistIt was used. Titan performed with similar response times as Neo4J and was chosen as most feasible for implementation for future versions of Manta.

4 Implemented and future research features

Based on customer demands, several features needed to be implemented into Manta database. First challenge was implementing temporal dimension, followed by multiple views on data (multiple hierarchies). Both topics became foundation for student's thesis (see [4] and [5]). Not yet implemented, but demanded features are i.e. incremental data load, parallel load, transactions during data load or united API for internal database access.

4.1 Temporal dimension

Implementing graph database as main data storage allowed adding temporal dimension to track structural and data flow changes in time.

Current implementation holds tempora6l data on edges (for both structural and data flow paths) and are indexed with range index type. Each edge stores a two-tuple TRAN _ START, TRAN _ END. The first property holds a revision id associated with introduction of the edge, the second one holds revision id referring to the moment the edge (and appropriate node) was deleted or it contains the current revision id in the case it still exists. Revision ids are represented by integer to allow interval index to be used.

Revisions themselves are represented by individual node for each revision connected to revision root node (with special attribute for identification), but with no other connections to original graph. These revision nodes creates separated graph and stores additional metadata for each revision (i.e. revision number, revision timestamp, etc.).

Current implementation brings one important restriction, enforcing full data refresh and disallowing incremental loads, since whole principle would stop working. See [4] for detail discussion.

4.2 Multiple view/hierarchies

Data lineage software solutions (including Manta or SQLdep) usually supports visualization of physical layer only. Some customers are requesting possibility for implementing more than physical data view. Using other modelers, many customers drawn extensive diagrams describing their business processes. This diagrams can be linked to physical data representation in their databases. This customers request leads to more complex approach for internal metadata structure.

Since Manta implemented graph database, solution for this demand was researched. A prototype solution was analyzed and discussed in [5]. The principle is illustrated in Figure 1.

It presents two different models (views) and their mutual relationships: Model A represents physical data structure in a particular technology (relational database in our example), while Model B represents a business view on the same metadata.

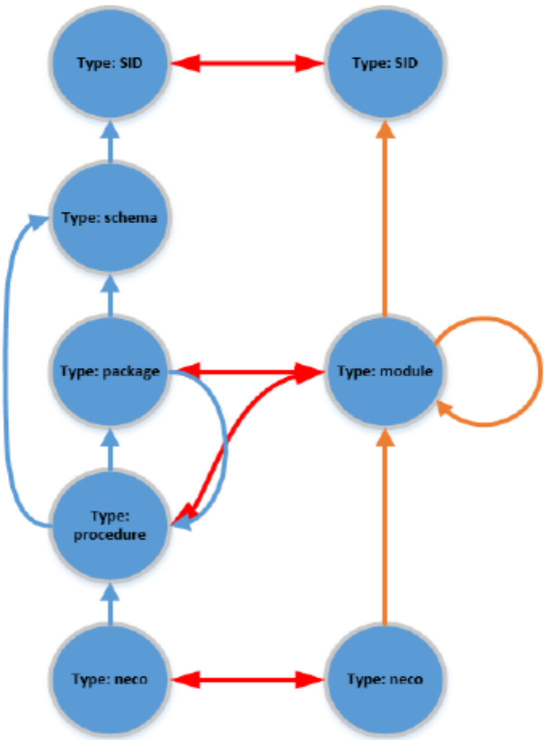


Fig. 1. Proposed solution for multiple hierarchies [5]

4.3 United API for internal database access

Since graph databases are still "young" discipline, headlong development is obvious. This situation complicates our research and brings more code rewriting than we're used to with relational databases. For example, at the beginning of our research, there were no unified graph database access API and switching to another graph database induced many code rewrites.

In last years, "*Blueprints*" for graph databases emerged and allows to use unified API to all graph databases which support it. Yet we decided to design and implemented our own API layer, which, is based on Blueprint. Our specific API layer better reflects the specific query needs for data lineage processing then universal Blueprint API. On the other hand utilizing of Blueprint guarantees easier change of internal graph database used in the project.

Implementation of this API layer is currently theme for Ondrej Bernát's thesis.

4.4 Transactrions during data load

Currently, we're not using transactions during data load process. Graph databases sometimes offer transactions, but implementation of this feature is not even near optimal yet. Without transactions, error during data load can cause disruption in temporal model and could result into wrong data lineage visualization.

For future internal database evolving, this issue needs to be solved, either by finding graph database that correctly and effectively implements transactions or we have to implement transactions ourselves.

4.5 Parallel and incremental data load

As mentioned before, there are several limitations in current solution, such as full data load/full refresh. Since temporal dimension is implemented based on [5], this concept needs to be revisited and updated to allow incremental data load. After eliminating this limitation, next step will emerge, parallel loading.

Customer databases are growing and even now, full data dictionary refresh by larger customer can take more than 15h. Since customers sometimes requires daily updates, incremental and parallel loadings needs to be solved. We've managed to get grant for this research (TAČR TH02010287) for time period 2017-2020 and solving this issue became main part of Karel Quast's dissertation work.

Acknowledgment

This work is partially supported by grants TAČR TH02010287 "Tools for governance of dynamic aspects of Big Data environment" and internal SGS grant number SGS17/211/OHK3/3T/1 Advanced research in software engineering.

References

1. Ling Liu, M. Tawer Oszu (editors): Encyclopedia of Database Systems. ISBN: 978-0-387-35544-3 (Print) 978-0-387-39940-9 (Online).
2. Robert Ikeda and Jennifer Widom. Data lineage: A survey. Technical report, Stanford University, 2009.

3. Michal Valenta: Návrh datového úložiště projektu Nástroje pro automatizaci Quality Assurance rozsáhlých Business Intelligence systémů a datových skladů. První výroční zpráva projektu TAČR. 2014.
4. Petr Holeček: Temporální data v grafové databázi v projektu Manta. Diplomová práce. České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2015.
5. Michal Peroutka: Optimální struktura a indexy modelu metadatového úložiště v grafové databázi. Diplomová práce. České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2016.
6. Mária Bieliková, Ivan Srba (editors): 11th Workshop on Intelligent and Knowledge Oriented Technologies 35th Conference on Data and Knowledge. ISBN: 978-80-227-4619-9.
7. Robert Ikeda, Hyunjung Park, and Jennifer Widom. Provenance for generalized map and reduce workflows. In Proc. of CIDR, January 2011.
8. Dionysios Logothetis, Soumyarupa De, and Kenneth Yocum. 2013. Scalable lineage capture for debugging DISC analytics. In Proceedings of the 4th annual Symposium on Cloud Computing (SOCC '13). ACM, New York, NY, USA, Article 17, 15 pages.

Parallel Itemset Mining Algorithms – an Overview

Adam Kaspar, Vojtech Kotik, and Jan Platos

Department of Computer Science, Faculty of Electrical Engineering and Computer Science, VSB – Technical University of Ostrava
17. listopadu 15, 708 33, Ostrava-Poruba, Czech Republic
{adam.kaspar, vojtech.kotik, jan.platos}@vsb.cz

Abstract. This paper is an overview of itemset mining algorithms and the published parallel approaches. Besides the state of the art in this area, we present also our results with the optimized Apriori variant implemented using OpenMP approach. Our implementation is discussed in detail with all used aspects.

Keywords: data mining, apriori, big data, algorithm, computations, parallelization

1 Introduction

Big data is a term that describes the large volume of data both structured and unstructured that inundates a business on a day-to-day basis. But its not the amount of data thats important. Its what organizations do with the data that matters. Big data can be analyzed for insights that lead to better decisions and strategic business moves.

Currently, we have a lot of computational algorithms. Most of them are based on Apriori algorithm. Data mining algorithms are divided into two based branches - string mining and itemset mining.

String mining typically works with a closed alphabet for elements that occurs in a sequence. String sequence could have various length. A typical use case of string mining could be found in conjunction with DNA research. String mining is here used for examination of known patterns of DNA sequences[1].

Itemset mining is useful for problems, related to sequence mining. A typical use of itemset mining is in marketing applications for discovering regularities between items on shopping lists.

Among commonly used algorithms for string mining belongs PrefixSpan, FreeSpan and for itemset data mining belongs FP-Growth, FP-Tree, Eclat, Max-Miner. Some of them I will describe in this paper.

2 Association rule mining

Association rule learning is a rule-based machine learning method for discovering interesting relations between variables in large databases. It is intended to identify strong rules discovered in databases using some measures of interestingness[2]. An example of these strong rules is association rules for discovering regularities between products that appears on the receipt. For example, let's have rule {apple, orange} ⇒ {strawberry}. This rule indicates, that if the customer buys apple and orange together in one purchase than the most likely the customer will also have strawberry in his shopping cart.

2.1 Apriori algorithm

Apriori is a base algorithm, belonging into frequent item set algorithms. The main idea is based on searching of larger and larger sets of items as those item sets occurs sufficiently often in the database[3].

The algorithm was designed by Agrawal and Srikant in 1994. Apriori works over databases, with transactions. These transactions could be for example collections of shopping cart items, bought by customers.

Basic functionality. Apriori works with "bottom-up" approach, it means, that at each step of the algorithm are frequent subsets extended about the new item. This step is called as candidate generation. After the candidate generation, resulted groups of candidates are tested against the data in the database. The algorithm ends, when no extension of frequent subsets is found. The entire scheme of the apriori algorithm is visible on the figure 1.

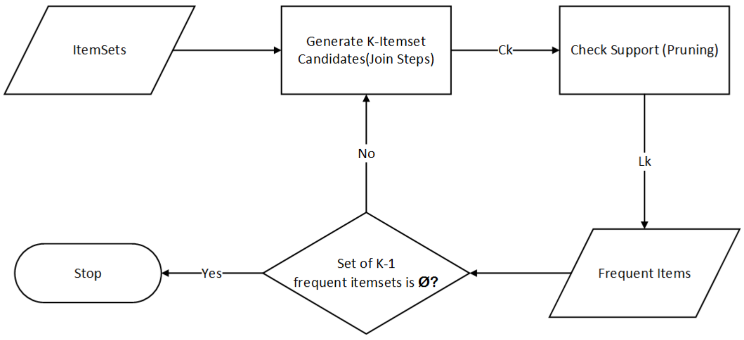

ItemSets
Generate K-Itemset Candidates(Join Steps)
Ck
Check Support (Pruning)
Lk
No
Frequent Items
Set of K-1 frequent itemsets is Ø?
Yes
Stop

Fig. 1. Apriori algorithm scheme.

Let's have the database that is depicted in Table 1.

Each row in this database represents one transaction at the supermarket and intersection of the row and column represents one item of each transaction.

As we can see, each transaction contains a pair of apple and orange, it means, that 100% transactions contain this pair. One-half of all transactions contain items { apple, orange, strawberry } or { apple, orange, lemon }.

The transactions may be stored in a database using the binary matrix as is depicted at Table 2.

tid	content
1	apple, orange, strawberry
2	apple, orange, lemon
3	apple, orange, strawberry
4	apple, orange, lemon

Table 1. An example dataset

tid	apple	orange	strawberry	lemon
1	1	1	1	0
2	1	1	0	1
3	1	1	1	0
4	1	1	0	1

Table 2. An example dataset represented using binary matrix

The name of the items will be from know denoted using its first letter (a, o, s, l).

Definition 1. *The support of an itemset I, $sup(I)$, is defined as the fraction of the transactions in the database $T = T_1, \ldots T_n$ that contain I as a subset.*

Definition 2. *An itemset I is called a Frequent Itemset when the support of an itemset is at least minsup.*

In our example, we define the *minsup* to 50%.

At the next step, Apriori algorithm has to count support value for each item in the database. To achieve this result, Apriori has to scan entire database. The itemsets are generated incrementally from single items to tuples, triples, etc. when possible. Only when all subsets of itemset are frequent an itemset is investigated. The list of all tested and found frequent itemsets are depicted in Table 3.

As we can see, the minimal support for each single itemset is equal to 2. The Itemset { s, l }, doesn't meet minimal support, it means, that this combination won't be contained in any future items.

Triplets, which passed minimal support are { a, o, s } and { a, o, l }. The algorithm ends at this moment because there is no any other items in the next generation of k-itemset.

2.2 Parallel implementation of Apriori

Basically, from performance point-of-view, when support is less, then more time is taken to run the program. This slowdown is caused mainly with multiple scans of the database. On the other hand, when the value of support is increasing, then time to run the Apriori will be less.

Item	Support
a	100%
o	100%
s	50%
l	50%
a, o	100%
a, s	50%
a, l	50%
o, s	50%
o, l	50%
s, l	0%
a, o, s	50%
a, o, l	50%

Table 3. Frequent itemsets found by an Apriori algorithm

One of the possible solutions, how to accelerate the Apriori is to parallelize the algorithm, for example with Open MPI library. Key factors of Apriori parallelization are:

- Divide the datasets.
- Each processor P_i will have its data set subset D_i.
- Each processor P_i reads the values of the data set from a large flat file.
- Each processor do a calculation of count of item sets in its particular processing unit.

Basic functionality. In the beginning, support is delivered to the first processor. This processor then distributes the value of the support to the other processors. Each processor is also responsible for initial item sets generation from the input data. The data is then divided between different processors. Is up to each processor to make decisions, when to terminate or continue with computation.

Then, each processor P_i develops the complete C_k, using itemset, created at the end of pass k-1. Processor P_i computes local support for candidates in C_k. After that, each processor sends local support to master processor to create global C_k support counts. After that, each processor computes L_k from generated C_k.

Parallel implementation of Apriori has been tested against the different count of processors and with different levels of support[4].

As you can see on the figure 2, for smaller support, the time for computation is longer than time for higher value of support. Also, with higher count of processors, the time for computation is shorter.

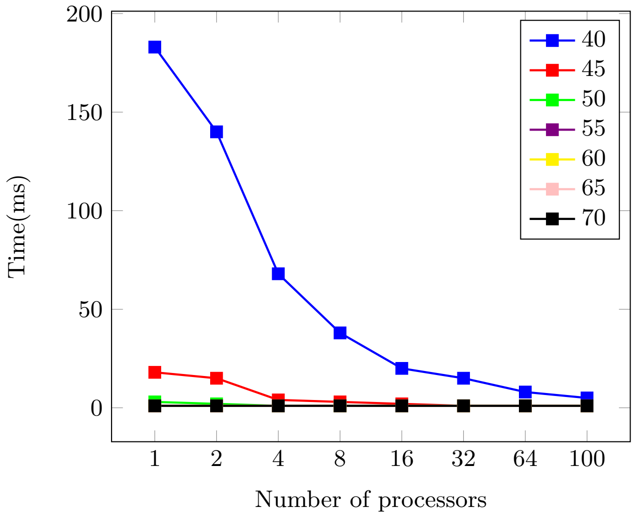


Fig. 2. Parallel Apriori performance comparison graph with different levels of support[4].

2.3 Eclat

Eclat (Equivalence Class Transformation) algorithm belongs among frequent itemset mining algorithms. This algorithm was first introduced by Mohammed J. Zaki, Srinivasan Parthasarathy, Wei Li and Mitsunori Ogihara in 1997 [6].

Basic functionality. The basic idea is based on depth-first search mining using the set intersection. The algorithm is suitable for sequential and also parallel execution. In the beginning, algorithm expects vertical database as his input. After that, Eclat works with tid-set of each item and verifies, that each itemset-tidset pair $\langle X, t(x) \rangle$ with all the others pairs $\langle Y, t(Y) \rangle$ to generate new candidates N_{xy}. If the candidate N_{xy} is frequent, then this candidate is added to the set P_x. Then, recursively, it finds all the frequent itemsets in the X branch[9], see an pseudo-implementation on listing 14.

Main advantage of Eclat is, that this algorithm have very fast support counting. On the other hand, among main disadvantage of this approach is that intermediate tid-lists may become too large for memory[10].

2.4 FP-Growth algorithm

The Frequent Pattern Growth (FP-Growth) algorithm was firstly proposed by J.Han in 2000[5]. In his work, Han proved, that FP-Growth is more efficient than other frequent pattern methods, like for example well-known Apriori[3] algorithm or Eclat[6].

Algorithm 1: Eclat(Transactions: T, Minimum support: $minsup$)

```
1  begin
2      Initialize set $\mathcal{ET}$ to single tuple $(Null, T)$ ;
3      while any node in $\mathcal{ET}$ has not been examined do
4          Select an unexamined tuple $(P, T(P))$ from $\mathcal{ET}$ for examination;
5          Generate candidates extensions $C(P)$ of each tuple $(P, T(P))$;
6          Determine frequent extension $F(P) \subseteq C(P)$ by support counting of
             individual items in smaller projected database $T(P)$;
7          Remove infrequent items in $T(P)$;
8          foreach each frequent item extension $i \in F(P)$ do
9              Generate $T(P \cup \{i\})$ from $T(P)$;
10             Add $(P \cup \{i\}, T(P \cup \{i\}))$ into $\mathcal{ET}$;
11         end
12     end
13     return set of tuples $\mathcal{ET}$
14 end
```

FP-Growth algorithm is one of the possible ways, how to find frequent itemsets. Concretely, it uses a divide-and-conquer strategy. As a base algorithm is used a special data structure, named as the frequent-pattern tree (FP-tree). The algorithm works as follows: first of all, it creates FP-Tree. This tree structure represents frequent items. After this step, the FP-Tree is divided into a set of conditional databases. Each database is then mined separately.

The main advantage of this algorithm against Apriori is that FP-Growth reduces the search costs looking for short patterns recursively and then concatenating them in the long, frequent patterns, offering good selectivity.

From performance reasons, for large databases, it isn't possible to hold entire FP-Tree in the main memory. The solution is to divide the database into a set of smaller databases and then construct an FP-Tree from each smaller database[7].

FP-Tree structure. The structure of FP-Tree consists of following elements. The tree contains root node, which is labeled as a null element. This element references child elements, named as item-prefix subtrees[5].

Each node in item-prefix subtree consists of these fields:

– Item ID
– Count: the number of transactions represented by the portion of the path reaching the node.
– Node-link: links to the next node in the FP-tree carrying the same item-name, or null if there is none.

After the FP-Tree is constructed, then FP-Growth algorithm is called. See figure 3, where the entire FP-Tree structure is sufficiently explained.

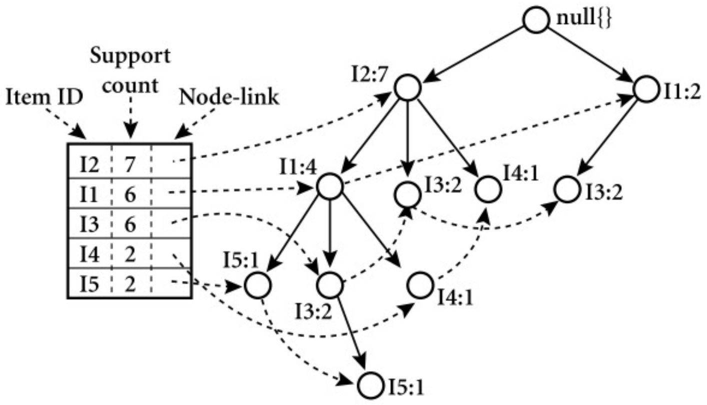


Fig. 3. FP-Tree structure[7].

FP-Growth performace comparison. FP-Growth algorithm was mined against 2-itemsets with different levels of support. There are also represented other mining algorithms, like Apriori, HI-Apriori, DHP.

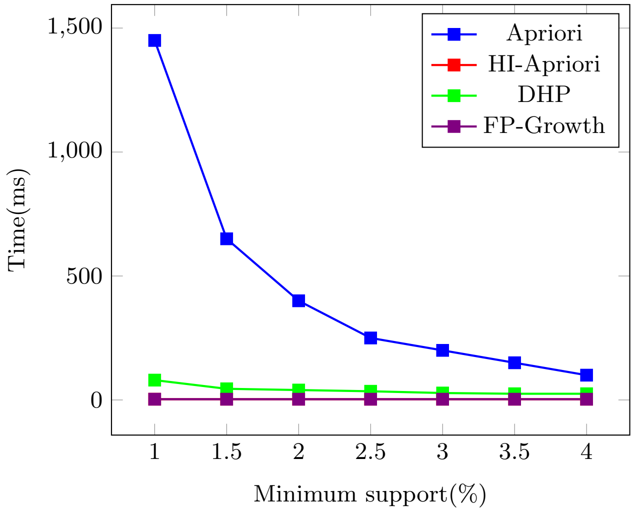


Fig. 4. FP-Growth performance comparison graph[8].

As we can see on the figure 4, there is a significant deviation for Apriori algorithm, when the level of minimal support is less than 2%. On higher levels of support, the performance of algorithms is almost the same[8].

3 Parallel implementation of Apriori algorithm

This section contains a description of our preliminary results in Apriori algorithms that were parallelized using OpenMP technology. Our implementation is an adaptation of the Eclat approach with the vertical organization of transactions. As was mentioned earlier, the Eclat algorithm works in iterations, and three phases are processed in each iteration.

1. Candidate generation phase where a list of candidates of length k is generated from the frequent itemsets of length $k-1$.
2. Computation of the support phase where support is computed for each generated candidate.
3. Pruning phase where the candidates with support lower than a min-support are removed.

The experiments show that he bottle-neck of the algorithm is the second phase, where the support is computed. We used parallelization of this phase using the OpenMP.

3.1 Results

In our experiments, we used a Windows based system with 2x Intel Xeon E5-2680 v2 @ 2.8GHz 10 core, 768 GB RAM. The dataset used in our experiments is Supermarket dataset from the Weka software. We tested our algorithm on two version of a dataset. One has 925,400 transactions, and the second has 4,627,000 transactions and 216 features. The results are depicted in Figure 5.

As may be seen, the second phase is the most important, because when the number of threads is increased by factor two, the time decreases with the same factor. This preliminary result shows that the parallelization is possible, but it is limited by the amount of memory. The tested dataset is small even in its full variant. We set the minimum support to 0.1 and the maximum speed-up of 90.3%.

4 Conclusion and future work

The itemset mining and associative rule mining algorithm is very useful tool in data analysis. More exactly, this tools may discover the hidden relations between a set of items, such as shopping carts, medical results, weblogs, etc. In these days, where such data volume is growing and becomes millions, and billions of transactions with thousands feature it is necessary to design efficient parallel version of the algorithm. In this paper, we presented a review of the published parallel version of the classical algorithm as well as preliminary results of our applications. The algorithm is efficiently parallelized, but it has to be modified to maintain reasonable memory requirements. That will be the future work in this task.

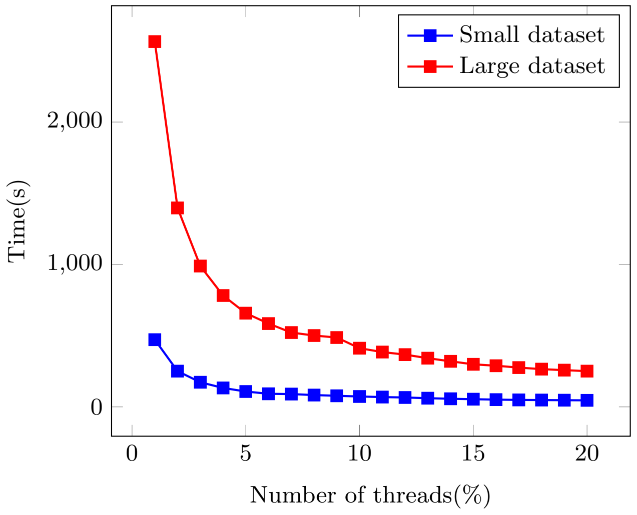


Fig. 5. Processing time of parallel version of Eclat algorithm on small (S) and large (L) dataset with different number of threads used.

Acknowledgment

This work was supported by the project SP2017/100 "Parallel processing of Big Data IV", of the Student Grant System, VŠB-Technical University of Ostrava.

References

1. Mohamed Abouelhoda, Moustafa Ghanem: String Mining in Bioinformatics. Springer Berlin Heidelberg, Berlin (2010)
2. Gregory Piatetsky-Shapiro, William J. Frawley: Discovery, Analysis and Presentation of Strong Rules. AAAI/MIT Press, Cambridge, MA (1991)
3. Rakesh Agrawal, Ramakrishnan Srikant: Fast Algorithms for Mining Association Rules. IBM Almaden Research Center, San Jose, CA (1991)
4. Sujith Mohan Velliyattikuzhi: Parallel implementation of Apriori algorithm and association of mining rules using MPI. Fall, Washington (2012)
5. J. Han, H. Pei, and Y. Yin: Mining Frequent Patterns without Candidate Generation. ACM Press, New York, NY, USA (2000)
6. M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li: New Algorithms for Fast Discovery of Association Rules. AAAI Press, Menlo Park, CA, USA (1997)
7. Jiawei Han and Micheline Kamber: Data Mining: Concepts and Techniques. Morgan Kaufmann (2006)
8. Shankai Yan and Pingjian Zhang: A Fast Association Rule Mining Algorithm for Corpus. Conference: 2013 International Conference on Intelligent Systems and Knowledge Engineering, At Shenzhen, Guangdong, Volume: 279 (2014)
9. Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li: New Algorithms for Fast Discovery of Association Rules. University of Rochester, Rochester NY (1997)

10. Jiawei Han, Vipin Kumar: CIS527: Data Warehousing, Filtering, and Mining Lecture. CIS, Temple University (2004)

Framework and Automated Prioritization Procedure for Model-Based Testing of Automotive Distributed Systems

Lukáš Krejčí and Jiří Novák

Czech Technical University in Prague, Faculty of Electrical Engineering, Techická 2, 166 27 Prague 6 - Dejvice, Czech Republic
{krejclu6, jnovak}@fel.cvut.cz

Abstract. The paper presents an optimized framework for model-based testing of automotive distributed system and a method of automatic assignment of testing priorities used within the framework. The proposed method, envisioned for integration into existing, currently developed model-based testing tool, utilizes classifiers in order to assign testing priorities automatically to specific parts of the tested system. The paper also introduces a set of supplemental data enriching the modeling language that are exploited by the proposed method during the classification process. It is shown, that advantages of the presented approach, such as low requirements for the testing operators' insight, are valuable for the automotive distributed systems testing process.

1 Introduction

The Model-Based Testing (MBT) is a popular approach to automated testing, which utilizes an abstract model of a System-Under-Test (SUT) in order to generate a testing sequence (i.e. test case). Alternatively, the testing procedure can be driven in real time using the MBT approach. The MBT can be applied even to complex software or hardware SUTs, such as automotive distributed systems.

Typically, the automotive industry uses a testing process composed of three distinct parts: the mandatory standard assessments, the specific test cases, and integration testing. The mandatory standard assessments are given by various international, national or organization safety standards and are thus unavoidable. The prearranged specific test cases are often based on the organization know-how and are therefore desirable, as they can uncover specific corner-case faults. The purpose of integration testing is to evaluate the automotive system as a whole since such testing can discover possible failures caused by distributed systems interconnection. Such failures characteristically originate from the inconsistent interpretation of the system specification among the subsystems suppliers, which may result in incorrect implementation and consequently in malfunction of multiple subsystems. The integration testing is performed regularly during entire vehicle lifecycle, usually during development and after every subsystem change.

Presently, the test cases for integration testing have to be designed manually by testing designers. Process of manual design of test cases can be, nevertheless, dubious, since it depends on individual test designers and therefore is vulnerable to subjective errors. In order to eliminate such disadvantage, design of test cases for integration testing can be automated using the MBT principles. However, as the automotive distributed systems are reactive, real-time and parallel, it was necessary to develop the appropriate tool, that would fit such requirements.

Accordingly, the Aim of this paper is to propose a new, optimized framework for MBT of automotive distributed systems and particularly present its automated testing priority assignment procedure. This procedure, referred as Automated Pinpointing, is responsible for assigning priorities to parts of SUT, which allows the proposed framework to generate most appropriate test cases. Thus, the Automated Pinpointing procedure is a Model-Based approach of test prioritization.

2 Background

The developed testing tool, called Taster, is based on diploma thesis [1] and described in the paper [2]. Since automotive systems are real-time and reactive, a modeling language used by this tool is based on the theory of timed automata, developed by UPPAAL team [3]. This modeling language allows describing modeled system as a network of Timed Safety Automata (TSA) bound by a set of variables.

The original theory of Timed Automata (TA) is described in [4]. The TSA, described in [5], differs from TA by usage of local invariant conditions that ensure automaton progress. Formal definition of a single TSA is following (described in [6]):

- A timed safety automaton A is a tuple $A = (N, l_0, E, I)$, where:
    - N is a finite set of locations (i.e. nodes),
    - $l_0 \in N$ is initial location,
    - $E \in N \times B(C) \times \Sigma \times 2^C \times N$ is the set of edges and
    - $I: N \to B(C)$ assigns invariants to locations.
- We shall write $l \to g, a, r, l'$, when $(l, g, a, r, l') \in E$.
- A local invariant is a constraint $x < n, x \leq n$, where $n \in \mathbb{N}$.

Informally, TSA is an oriented graph containing states (one of them is initial) and transitions between them. Transitions are labeled by guard condition enabling its execution.

Each TSA in an SUT model is referred to as template and usually represents a standalone model of some particular SUT subsystem. Since the MBT approach requires a model of the SUT environment, one TSA usually serves as environment model.

Since it is essentially impossible to perform its complete evaluation due to significant complexity of a typical automotive distributed SUT, only some parts of its state space can be tested. Therefore, it is important to generate as most appropriate test cases as possible. Numerous strategies utilizing various graph algorithms can drive the test sequence generation to achieve such goal. An example of such strategy is a generation of pseudorandom test sequences according to priorities manually inscribed in the model. In presence, this strategy is the only one supported by the Taster tool. While this strategy may provide valid results in several cases, its major pitfall lies in the manual assignment of testing priorities. Since the correct assignment of priorities is affected by multiple parameters, this approach requires significant insight from testing operators. Moreover, utilization of pseudorandom sequences can lead to suboptimal results in several cases (i.e. SUT with clustered state-space). Consequently, a new optimized testing framework for this tool was designed to reduce the mean time required for the failure detection and to lower the amount of insight required from testing operators. Fig. 1 shows its structure.

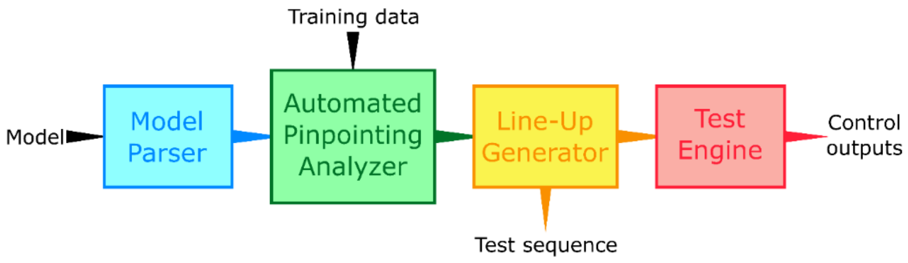


Fig. 1: The structure of proposed framework.

The framework is defined as a pipeline chain consisting of following blocks:

- The Model Parser, which loads a model from its XML-formatted file and converts it into an internal representation. Schema of XML file is compatible with the schema defined by UPPAAL and thus the models used by presented framework can also be formally verified using the UPPAAL model checker.
- The Automated Pinpointing Analyzer, which encases the procedure of labeling the model with testing priorities. This block takes parsed model as input and enriches it with testing priorities.
- The Line-Up Generator is responsible for generating the testing sequences. This block takes an enriched model as an input and generates the control outputs for the Test Engine according to chosen test generation strategy.
- The Test Engine, which directly executes given operations on the SUT using supported testing backend (i.e. NI VeriStand or Exam).

This paper focuses on the presentation of the labeling procedure encased in the Automated Pinpointing Analyzer block. This procedure, called Automated Pinpointing, utilizes classification of extra data stored in the model and is described in section 4.

3 Related Work

The MBT and test generation are still active research topics [7]. Currently, a variety of different approaches exist. The Model-Based Statistical Testing (MBST) is an MBT approach utilizing the empirically obtained behavioral models. Authors of [8] describe the usage of MBST principles on embedded systems. Due to the purpose of integration testing, the MBST approach does not seem to be viable. However, the MBST usage could be reconsidered in future, e.g. to discover failures caused by unexpected user behavior.

As the presented approach uses a testing priorities for driving the test sequence generation process, it can be interpreted as the Risk-Based Testing (RBT). The paper [9] describe the usage of the MBST and RBT principles for testing of railway control system. Authors focus on the feasible method of test sequences generation, and the problem of marking the critical parts of the system is left to the future research.

Authors of [10] investigate utilization of MBT principles on the testing of graphical user interface and usage of classifiers for identification of infeasible test cases. Method presented in this paper utilize classifiers as well, however, for a different purpose.

In all papers described above (and even in commercial tools, such as [11]), the critical parts of SUT have to be manually marked in the model. As stated in section II, the manual assignment can be dubious, since it depends on various subjective factors that test cases designers have to take into account. Consequently, in order to avoid this potentially dubious process, this paper proposes a method that assigns testing priorities automatically.

Since the proposed approach utilizes automatic assignment of testing priorities, it is related to the area of test prioritization. Authors of [12] describe a method of automated coverage-based test prioritization utilizing artificial neural networks. Though method presented in this paper utilizes artificial neural networks as well, the main difference between it and method described in [12] lies in the fact that in the case of [12], the prioritization of test case is performed after the test case is generated. As mentioned in section III, proposed framework initially assigns priorities to parts of SUT and afterward generates test case according to assigned priorities. Additionally, prioritization used in proposed framework is not coverage-based, but is model-based instead.

4 Automated Pinpointing Procedure

As revealed in section II, the purpose of Automated Pinpointing procedure is to enhance the provided system model with testing priorities. In the utilized modeling language (i.e. UPPAAL defined networks of TSA), functionalities of an SUT are bound to the transitions and templates in the model. Consequently, the procedure assigns testing priorities to them.

The procedure iterates through all transitions and templates in the model and assigns a priority to each of them. Nevertheless, in order to allocate the priorities in a viable manner, additional extra data providing further insight to the SUT are required.

The proper allocation of testing priority depends on a plethora of parameters, including both objective and subjective factors. Since the automotive systems consist of numerous safety-critical subsystems, the safety-impairment factor of particular functionality should be taken into account. Certainly, if some particular SUT functionality is safety-critical, it should have higher testing priority than a non-critical one. Similarly, it is vital to allocate higher testing priority to systems that are not necessarily safety-critical, but significantly impact overall car user experience. Moreover, if a particular functionality of the SUT is expected to be a source of a failure, it is desirable to assign a high testing priority to it. Failure of a certain SUT functionality can be predicted using several objective factors that reflect its complexity, reliability of its back-end, and the amount, severity and authors of changes recently made to it. The safety importance and user experience impairment aspects are vastly subjective, as they depend on an individual view of manufacturer or user. From the other hand, factors that can be used for failure prediction are objective.

The Automated Pinpointing procedure utilizes two categories of extra data inscribed into a model on the transition and template level. The fault indicators are used for prediction of particular transition or template failure. Failure prediction is afterward combined with the experience aspects to obtain transition or template testing priority, which is interpreted as an integer in the range from 1 to 10. This workflow is depicted in Fig. 2.

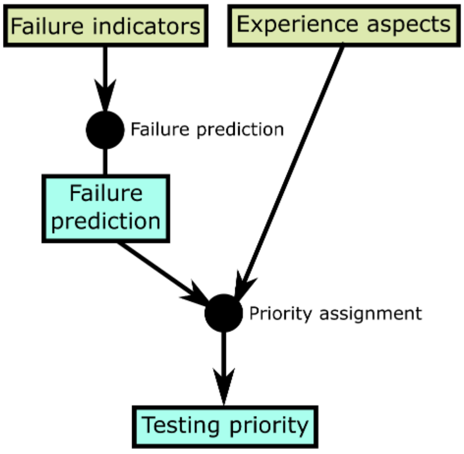


Fig. 2: Testing priority assignment workflow.

Details about failure prediction and priority assignment procedures follow in sections 4.1 and 4.2.

4.1 Failure Prediction

As the section IV declares, assignment of testing priority depends on failure prediction. In order to enable prediction of failure of particular functionality, the following failure indicators have been introduced as additional transition and template extra data:

- The *Modification Amount* ratio (real number in the range from 0 to 1) representing the amount of changes made to the particular functionality since the last evaluation. This value needs to be updated after each update of the functionality.
- The *Modification Severity* index (integer in the range from 1 to 10) representing the severity of changes made to the particular functionality since the last

evaluation. This value needs to be updated after each update of the functionality.

- The *Authors ID* value (integer) representing the authors of changes made to the particular functionality since the last evaluation. This value is obtained as SHA-256 of a textual string containing author's full name. In the case of multiple authors, this value is obtained as SHA-256 of a textual string containing hash codes of authors' full names separated by a semicolon. This value needs to be updated after each update of the functionality.
- The *Control Units Count* value (unsigned integer) representing a total number of control units involved in the functionality. This value needs to be updated only when a total number of participating control units is changed in the SUT.
- The *Manufacturers Count* value (unsigned integer) representing a total number of manufacturers of control units involved in the functionality. This value needs to be updated only when a total number of manufacturers of participating control units is changed in the SUT.
- The *Back-End Reliability* ratio (real number in the range from 0 to 1) representing the reliability of a back-end (i.e. hardware and lower software layers) of particular functionality obtained from the testing of low-level parts of the SUT.

The rationale behind the *Modification Amount* and *Modification Severity* extra data is intuitive. Certainly, changes made to the functionality can cause its failure. Both these values must be updated each time a change to particular functionality is made.

Involvement of certain authors of functionality changes can, when combined with specific development patterns, indicate possible failure. Therefore, the *Authors ID* extra data has been introduced. Similar to the *Modification Amount* and *Modification Severity*, this value must be updated after each change to particular functionality.

The values of the *Control Units Count* and *Manufacturers Count* extra data together reflects the overall complexity of particular functionality since the higher complexity of functionality may make its failure more likely. Since the numbers of co-operating control units and their manufacturers characteristically do not change often, these values will typically be updated only occasionally.

Additionally, the functionality with unreliable back-end is likely to cause a failure. Thus, the *Back-End Reliability* extra data has been introduced. Its value directly reflects the results of back-end testing (i.e. the results of independent control units testing) and therefore must be updated whenever new back-end testing results are available.

Clearly, the failure prediction can be taken as a classification problem, where the input vector consists of the failure indicators and output is an indication of possible failure. More formally:

- $F = \{0, 1\}$ is the set hidden states,
- X, the features space, is given by introduced extra data,
- $D = F$, the set of possible decisions.

Finding the optimal decision strategy for this particular decision problem is problematic. The failures of SUTs are in case of automotive distributed systems usually very rare and strongly depend on a SUT development environment (i.e. specific combination of manufacturer, programmer, tool or development methods used for the SUT development). Therefore, the prior probability of failure $p(F)$ cannot be properly obtained empirically. Moreover, the conditional probabilities of observations given the hidden state $p(X|F)$ have unknown probability distribution, which is also strongly affected by a SUT development environment and so they cannot be obtained empirically as well. Because the generative approach, i.e. finding the probability model itself, is problematic, the discriminative approach of Artificial Neural Network (ANN; their overview described in [13]) has been chosen as a classification method suitable for this task.

Since the actual values of input vectors strongly depend on a specific combination of manufacturer, programmer, tool or development method used for the functionality modification or utilized back-end, the feature space is expected to be heavily clustered and not linearly separable, so a simple solution with a single perceptron neuron is infeasible [14]. Therefore, two neural networks have been proposed for this task.

The structure of first proposed ANN is shown in Fig. 3.

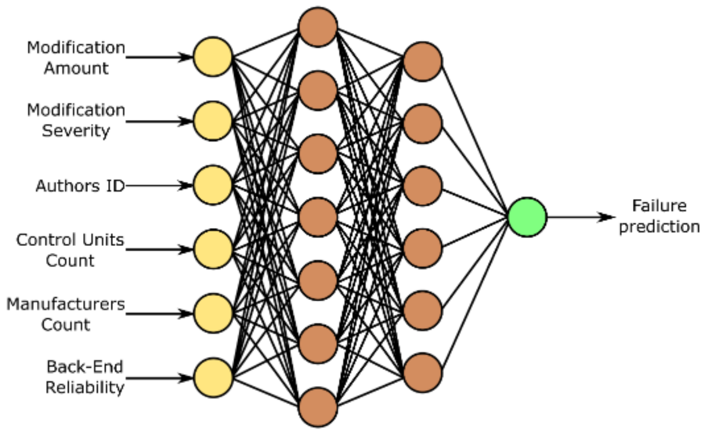


Fig. 3: Structure of multilayer perceptron feedforward network.

The first proposed ANN is a multilayer perceptron feedforward network composed of two hidden layers; the first layer consists of seven neurons and seconds layer consists of six neurons. All neurons use the sigmoid function as an activation function. The input weights of all neurons are initialized randomly. The ANN uses supervised learning process utilizing standard backward propagation procedure applying the provided input vector and the actual test result as expected output. The inference process is realized by the forward propagation.

While this approach's advantage lies in the relative simplicity of its structure, it may struggle with adaptation to specific development patterns (i.e. clusters in the feature space). Therefore, second ANN was proposed. Its structure is depicted in Fig. 4.

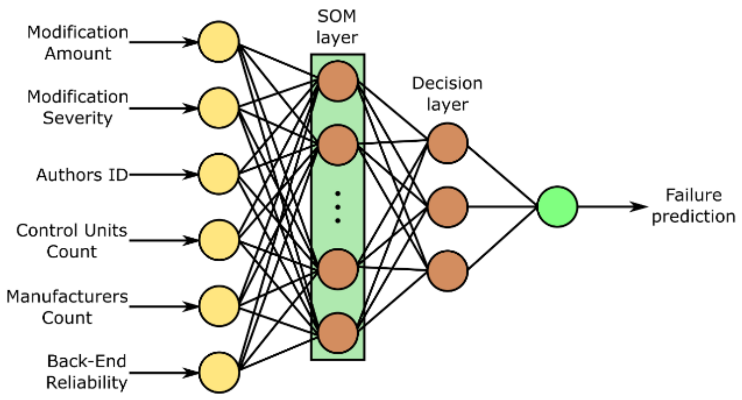


Fig. 4: Structure of SOM-based network.

The second ANN is, similarly to the first proposed ANN, composed of two hidden layers. However, this ANN utilizes a Self-Organizing Map (SOM; described in [15]) as the first layer and a single-layer perceptron feedforward network as the second layer. The SOM layer is used for quantization of input vector and thus contains 4096 neurons initially uniformly distributed among feature space as a 6-D mesh. Each neuron of SOM layer uses the Euclidian distance as an output function. The decision layer is designed as a single-layered ANN composed from 3 neurons with sigmoid activation function and randomly initialized weights. During the learning process, the weights of SOM and decision layer are adjusted using standard approaches, i.e. supervised learning of the decision layer by backward propagation and unsupervised learning of the SOM layer.

During learning process, the structure of SOM layer will be modified according to the training data structure. Consequently, usage of the SOM layer allows this ANN to adapt more smoothly to the specific SUT development patterns applied by particular automotive systems manufacturer. However, the disadvantage of the SOM approach is a significantly higher number of neurons.

4.2 Priority Assignment

As mentioned in section IV, proper assignment of testing priority also depends on several subjective factors. Therefore, following experience aspects have been introduced into transition and template extra data:

- The *Safety Impairment* index (integer in the range from 1 to 10) representing the impact of the functionality failure on the vehicle safety.
- The *User-Comfort Infringement* index (integer in the range from 1 to 10) representing the impact of the functionality on the comfort of the vehicle users.

The rationale behind these aspects is evident. The higher each index is, the more severe effect of the failure on the vehicle safety or users comfort are. Therefore, the higher index together with the possible prediction of failure implies higher testing priority. Moreover, their values do not need to be updated. As these aspects are subjective, their appropriate values have to be obtained empirically.

The process of priority assignment can be, similarly to the process of failure prediction, interpreted as a process of classification, where the classifier assigns a priority according to the input feature vector composed of the *Safety Impairment* index, *User-Comfort Infringement* index and the failure prediction. Undoubtedly, from the perspective of a single priority, such feature space is linearly separable and thus simple perceptron classifiers can be utilized for this task [14]. Fig. 5 shows a feature space linearly separated by hyperplanes into diagonal zones corresponding to priorities.

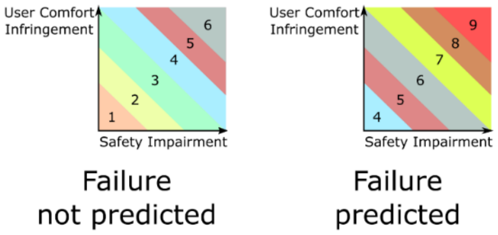


Fig. 5: Priority assignment feature space separation.

Since there are ten possible testing priorities, nine different perceptron classifiers are utilized, each having a fixed hyperplane associated with it. Fig. 6 depicts schema of the classification structure.

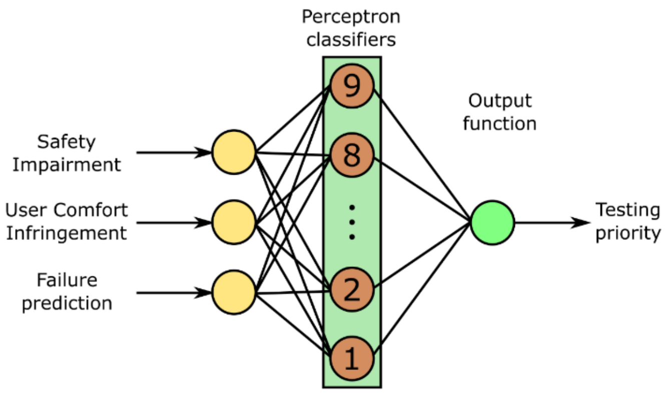


Fig. 6: Priority assignment classification structure.

The classification function of each perceptron returns 1, if the input vector lies below its associated hyperplane, and returns 0 otherwise. The output function of classification structure is defined as following.

$$f_{out}(y_1, \dots, y_9) = 10 - \sum_{i=1}^{9} y_i$$

Output of perceptron classifier assigned to i-th priority is denoted as y_i. The output function counts the outputs of separate perceptron classifiers and subtracts this sum from 10, which is then used as testing priority for given transition or template.

Since the testing priority is not objectively measurable, the priority assignment procedure does not use learning and utilizes a fixed configuration instead.

5 Conclusions and Future Work

In this paper, a part of the new framework for the model-based testing of an automotive distributed systems is presented. The main focus of the paper lies in the presented model-based approach of testing priority assignment, called Automated Pinpointing procedure. The Automated Pinpointing procedure exploits the SUT model supplemented with the additional extra data to assign testing priorities to model's transitions and templates automatically. This process is useful for identifying the SUT parts that are most worthwhile to be tested.

Additionally, an appropriate set of extra data was introduced. The proposed system of transition and template extra data contains both subjective factors (i.e. experience aspects), as well as objective factors (i.e. fault indicators). As supplemental data take into account various factors influencing the testing priority, the proposed method significantly lowers overall testing operators' insight requirements, as most of the required model parameters can be obtained repetitively by automatic procedures (the objective factors) or once by empirical research (the subjective factors).

Last but not least, three classification structures were proposed. Two ANN for failure prediction and one based on perceptron neurons for priority assignment. Since priority assignment result cannot be objectively measured, the priority assignment classification structure has static, preconfigured, structure and parameters.

Presently, the limited version of presented framework is implemented into developed tool Taster that currently utilizes pseudorandom approach for the generation of the test sequence. Since this tool uses a modeling language based on networks of timed automata, the presented framework, Automated Pinpointing procedure and proposed extra data utilize it as well.

Future research will be focused on the optimization of the classification structures used within the priority assignment procedure. Since the procedure makes use of strongly subjective *Safety-Impairment* and *User-Comfort Infringement* indexes extra data, proper values of these parameters will be acquired by empirical research. Also, it is necessary to gather training data to evaluate proposed failure prediction ANN structures, select the most promising one and optimize it afterward. Furthermore, other classification approaches will be assessed, so the most optimal one could be found. For example, the rule-based methods seem to be promising for this purpose.

Additional research will be oriented on the methods for prediction of possible regression in particular functionality caused by influences originating in other, possibly similar, SUT functionality. Such information can be obtained using the static analysis principles from the SUT source code or can be provided by the system modeler. Afterward, such information can be exploited by the context-sensitive classifiers for more precise prediction of functionalities failures.

Finally, since the developed tool currently generates testing sequences pseudo-randomly, more optimal test generation strategies will be explored.

The proposed procedure and complete Taster tool is evaluated, thanks to the co-operation with Škoda Auto a. s., on the real automotive systems developed and manufactured by this company. Moreover, Škoda Auto a. s. supplies data from real automotive systems that are necessary for the procedure optimization. Presented methods are currently tested on a case study with trunk door control unit. The case study includes several environment models providing various user stimuli, such as locking and unlocking the car, opening and closing the trunk door and starting and shutting down the engine, as well as complex observer model. The observer model describes the correct behavior of trunk door control unit and its most important parts are the invariant checks. This case study should help with fine-tuning of used ANN structures, as well as with evaluation of explored test generation strategies in the future.

Acknowledgement

Research described in the paper was supported by the Grant Agency of the Czech Technical University in Prague, project Model-Based Testing methods for automotive electronics systems (grant No. SGS16/171/OHK3/2T/13) and by the support of Technological Agency, Czech Republic, program Centers of Competence, project No. TE01020020 Josef Božek Competence Centre for Automotive Industry.

References

[1] Tomáš Grus, “Implementation of Integration Testing Test Cases Generation Tool,” *Master’s Thesis*, CTU in Prague, 2014.

[2] Sobotka, J. and Novák, J., “Testing Automotive Reactive Systems using Timed Automata,” 2016, [unpublished].

[3] Behrmann, G., David and A., Larsen, K. G., “A tutorial on UPPAAL,” *In proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT'04).*

[4] Alur, R. and Dill, D., “Automata for modeling real-time systems,” *In proceedings of the Seventeenth International Colloquium on Automata Languages and Programming*, *443*(443), 322–335. doi:10.1007/BFb0032042, 1990.

[5] Henzinger, T. A., Nicollin, X., Sifakis, J. and Yovine, S., “Symbolic model checking for real-time systems,” *Information and Computation*, *111*(2), 193–244, doi:10.1006/inco.1994.1045, 1994.

[6] Bengtsson, J., Bengtsson, J., Yi, W. and Yi, W., “Timed automata: Semantics, algorithms and tools,” *Lecture Notes in Computer Science*, *3098*(316), 87–124, doi:10.1007/978-3-540-27755-2_3, 2004.

[7] S. Rösch, S. Ulewicz, J. Provost, and B. Vogel-Heuser, “Review of Model-Based Testing Approaches in Production Automation and Adjacent Domains—Current Challenges and Research Gaps,“ *J. Softw. Eng. Appl.*, vol. 08, no. 09, pp. 499–519, 2015.

[8] F. Böhr, “Model Based Statistical Testing of Embedded Systems,” *2011 IEEE Fourth Int. Conf. Softw. Testing, Verif. Valid. Work.*, pp. 18–25, 2011.

[9] Zimmermann, F., Eschbach, R., Kloos, J. and Bauer, T., "Risk-based Statistical Testing: A Refinement-based Approach to the Reliability Analysis of Safety-Critical Systems," *12th European Workshop on Dependable Computing, EWDC 2009*, Toulouse, France, 2009.

[10] Gove, R., Faytong, J., "Chapter 4 – Machine Learning and Event-Based Software Testing: Classifiers for Identifying Infeasible GUI Event Sequences," *Advances in Computers,* vol. 86, pp. 109–135, 2012.

[11] MaTeLo, http://www.all4tec.net/MaTeLo/homematelo.html.

[12] Belli, F., Eminov, M., and Gocke, N., "Prioritizing Coverage-Oriented Testing Process - An Adaptive-Learning-Based Approach and Case Study," In *proceedings of the 31st Annual International Computer Software and Applications Conference - Volume 02,* pp. 197–203, 2007.

[13] Aleksander, I. and Morton, H., "An introduction to neural computing," *International Thomson Computer Press*, 2nd edition, 1995.

[14] Minky, M. and Papert, S., "Perceptrons: An Introduction to Computational Geometry," *M.I.T. Press*, 1969.

[15] Kohonen, T., "Self-Organizing Maps," *New York : Springer-Verlag*, 1997.

Model Transformations via XSLT

Jakub Pavlát[1], Karel Richta[1], Tomáš Richta[2], and Vladimír Janoušek[2]

[1] [1] Czech Technical University in Prague, Faculty of Electrical Engineering, Dept. of Computer Science,
Karlovo nám. 13, 121 35 Praha 2, Czech Republic
{pavljak, richta}@fel.cvut.cz
http://cs.felk.cvut.cz/en/people/richta

[2] [2] Brno Institute of Technology, Faculty of Information Technology, Dept. of Intelligent Systems,
Božetěchova 2, 612 66 Brno, Czech Republic
{irichta, janousek}@fit.vutbr.cz
http://www.fit.vutbr.cz/~irichta/,http://www.fit.vutbr.cz/~janousek/

Abstract. Many present systems are conceived as a set of autonomous agents that communicate together in solving problems. Petri nets are commonly used for a specification of parallel systems. An interesting question is whether we can implement arbitrary system by a set of Petri nets agents. These agents could be specified by Petri nets, but such a description is not sufficiently user-friendly. Our idea is to define these agents by classical workflow models and then transform them into a set of Petri nets. Such transformation would support development of software systems, whose specification is based on classical workflow models, but the implementation is based on Petri nets. Each net of the resulting system is translated into a specific target representation called Petri Nets Byte Code (PNBC). These codes are interpreted by the special Petri Nets Virtual Machines (PNVM), which are installed on all nodes of the system, under the Petri Nets Operating System (PNOS). This paper deals with the one step of this process, which is an implementation of required transformations from workflow model into Petri nets with the help of XSLT.

1. Introduction

Many present computer driven systems are considered to be a set of autonomous agents that communicate together to solve problems. There are a number of systems that support the provision of such communication on the basis of specifications of the external behavior of agents and a description of their communication. System assembly of a set of agents is relatively well developed and orchestration of such set of agents can be assured and generated from its description by known tools. What is currently not developed and supported by automation is a creation of agents based on their specifications. They are usually implemented manually without the assistance of an adequate environment.

The purpose of our research is to develop methods and tools that can be used for the creation of autonomous agents from their specification. We investigate the specification of agents based on Petri nets (see [7,8]), because we suppose, that all nodes of the distributed system can be equipped by PNVM (Petri Net Virtual Machine [10]) running under special PNOS (Petri Net Operating System [9]). Thus equipped nodes can be programmed by uploaded Petri nets, and pose as agents of the whole orchestrated distributed system.

The definition of an agent behavior can be expressed by a classical workflow model ([2]). Such definition is user friendly, and relatively simple. There exist tools supporting a creation of such descriptions. For our purposes these descriptions have to be converted into Petri nets. Our research deals with such transformations, which convert classical workflow models into equivalent Petri nets. In this article we will try to define rules that specify such transformation. These rules can be implemented using XSLT (eXtensible Stylesheet Language Transformations [14]). For this purposes, the source, and also resulting model should be expressed in the XML format. We use EMF (Eclipse Modeling Framework [1,12]) format for source models, and the PNML language (Petri Net Markup Language [3,13]) for resulting Petri nets. The resulting net description in PNML can be interpreted by a Petri net virtual machine. All such particular agents can be joined with the organizational part of a system, and participate in the implementation of the required behavior.

2. The Source Specification Model

Mainstream software processes, like the Unified Process, propose UML class diagrams for modeling the integration of conceptual data models and activity diagrams for specifying workflows [1]. Then, these models are used as requirements specification to drive the subsequent development and integration of applications through analysis, design, and implementation activities. The alternative approach consists in the use of the workflow specification as an executable model interpreted by a workflow management system. In this paper, we will use the input specification in the form of a workflow.

As an example for an input model consider some autonomous system, e.g. heating system for a small house, see Fig. 1. This heating system consists of five components („Hall and stairway“, „Kitchen and dining room“, „Remote control“, „Boiler room“, and „Scheduler“). The whole system containing these five components is called „House 1“. Components have input and output ports, e.g. components „Hall and stairway“ has among other the input port „schedTemp1“, and the output port „temp1“. The output port „temp1“ of „Hall and stairway“ is connected to the input port of the „Scheduler“, etc. Components contain elements like „thermostat“, „knob“, whose description and structure is already predefined.

Such a description could be created by some CASE tool, and finally exported in the XML format, e.g. as an application of EMF. Look at the Fig. 2, which shows the small part of the representation of this heating system in EMF. This XML document is a valid document according to the XML schema of EMF, here „**`domotic.ecore`**“ (see [4]). This schema is designed for so called „Domotic Systems“ - set of technological components capable of performing functions that can be partially autonomous, programmed by the user, or even completely autonomous.

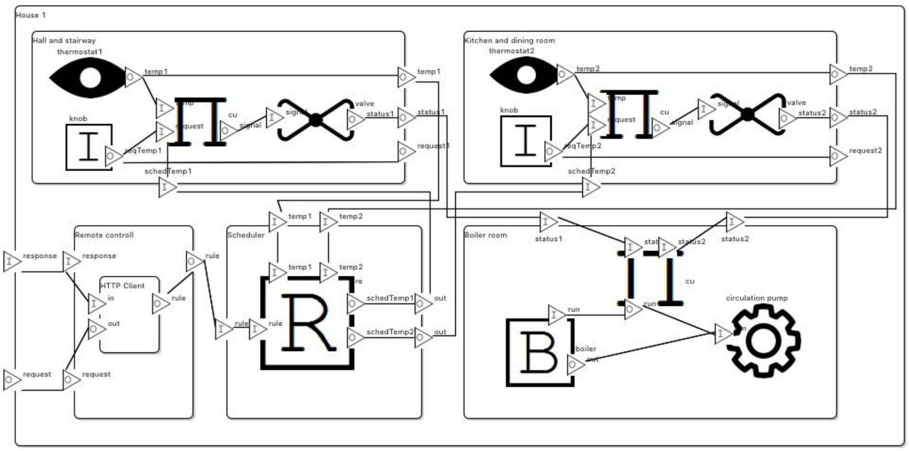


Fig. 1: Autonomous heating system

```
<?xml version="1.0" encoding="UTF-8"?>
  <domotic:Component xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:domotic="but-hib.domotic" xsi:schemaLocation="but-hib.domotic
../domotic.viewpoint/description/domotic.ecore" name="schedTemp1">
  <contains xsi:type="domotic:Component" name="House 1">
    <inputPorts name="response"/>
    <contains xsi:type="domotic:Component" name="Hall and stairway">
      <inputPorts name="schedTemp1"/>
      <contains xsi:type="domotic:Sensor" name="thermostat1">
        <outputPorts name="temp1"/>
      </contains>
      <contains xsi:type="domotic:Valve" name="valve">
        <outputPorts name="status1"/>
        <inputPorts name="signal"/>
      </contains>
      <contains xsi:type="domotic:UserInput" name="knob">
        <outputPorts name="reqTemp1"/>
      </contains>
      <contains xsi:type="domotic:ComputationalUnit" name="cu">
        <expression>signal = temp &lt; request</expression>
        <inputPorts name="temp"/>
        <inputPorts name="request"/>
        <outputPorts name="signal"/>
      </contains>
      <outputPorts name="temp1"/>
      <outputPorts name="status1"/>
      <outputPorts name="request1"/>
      <dataFlows name="df"
ports="//@contains.0/@contains.0/@contains.1/@inputPorts.0"/>
. . .
```

Fig. 2: The input model in EMF format

3. The Output Model - Petri Nets

Petri nets are widely used for the specification of problems, mostly in the parallel systems. The following formal definition is loosely based on [7, 8]. Many alternative definitions exist.

3.1 Syntax

A **Petri net graph** (called *Petri net* by some, but see below) is a 3-tuple (P,T,W), where:

- $P = \{ P_i \}$ is a finite set of *places*
- $T = \{ T_j \}$ is a finite set of *transitions*
- P and T are disjoint, i.e. no object can be both a place and a transition
- W: $(P \times T) \cup (T \times P) \rightarrow \mathrm{N}$ is a multi-set of arcs, i.e. it defines arcs and assigns to each arc a non-negative integer *arc multiplicity*; note that no arc may connect two places or two transitions.

The *flow relation* is the set of arcs: $F = \{(x,y) \mid W(x,y) > 0\}$. In many textbooks, arcs can only have multiplicity 1, and they often define Petri nets using F instead of W.

A Petri net graph is a bipartite multidigraph $(P \cup T, F)$ with node partitions P and T.

The *preset* of a transition t is the set of its *input places*: ${}^{\bullet}t = \{p \in P \mid W(p,t) > 0\}$; its *postset* is the set of its *output places*: $t^{\bullet} = \{p \in P \mid W(t,p) > 0\}$.

A *marking* of a Petri net (graph) is a multiset of its places, i.e., a mapping M: $P \rightarrow \mathbb{N}$. We say the marking assigns to each place a number of *tokens*.

A **Petri net** (called *marked Petri net* by some, see above) is a 4-tuple (P,T,W,M_0), where:

- (P,T,W) is a Petri net graph;
- M_0 is the *initial marking*, a marking of the Petri net graph.

3.2 Execution semantics

The behavior of a Petri net is defined as a relation on its markings, as follows. Note that markings can be added like any multiset:

$$M + M' =^{D} \{ p \rightarrow M(p) + M'(p) \mid p \in P \}$$

The execution of a Petri net graph G = (P,T,W), can be defined as the transition relation $\rightarrow_G$ on its markings, as follows:

- for any t in T:
  $M \rightarrow_{G,t} M' \Leftrightarrow^{D} \exists M'' : P \rightarrow \mathbb{N} : M = M'' + \Sigma_{p \in P} W(p,t) \wedge M' = M'' + + \Sigma_{p \in P} W(t,p)$
  $M \rightarrow_{G} M' \Leftrightarrow^{D} \exists t \in T : M \rightarrow_{G,t} M'$

In words:

- firing a transition t in a marking M consumes $W(p,t)$ tokens from each of its input places p, and produces $W(t,p)$ tokens in each of its output places p
- a transition is enabled (it may fire) in M if there are enough tokens in its input places for the consumptions to be possible, i.e. iff

  $\forall p{:}M(p) \geq W(p,t)$.

We are generally interested in what may happen when transitions may continually fire in arbitrary order.

We say that a marking M' *is reachable from* a marking M *in one step* if $M \rightarrow_G M'$; we say that it *is reachable from M* if $M \rightarrow^*_G M'$, where $\rightarrow^*_G$ is the *transitive closure* of $\rightarrow_G$; that is, if it is reachable in 0 or more steps.

For a (marked) Petri net $N = (P,T,W,M_0)$, we are interested in the firings that can be performed starting with the initial marking M_0. Its set of *reachable markings* is the set $R(N) =^D \{ M' \mid M_0 \rightarrow_{(P,T,W)}^* M' \}$.

The *reachability graph* of N is the transition relation $\rightarrow_G$ restricted to its reachable markings $R(N)$. It is the *state space* of the net.

A *firing sequence* for a Petri net with graph G and initial marking M_0 is a sequence of transitions $\sigma^{\rightarrow} = \langle t_{i1} \ldots t_{in} \rangle$ such that $M_0 \rightarrow_{G,ti1} M_1 \wedge \ldots \wedge M_{n-1} \rightarrow_{G,tin} M_n$. The set of firing sequences is denoted as $L(N)$.

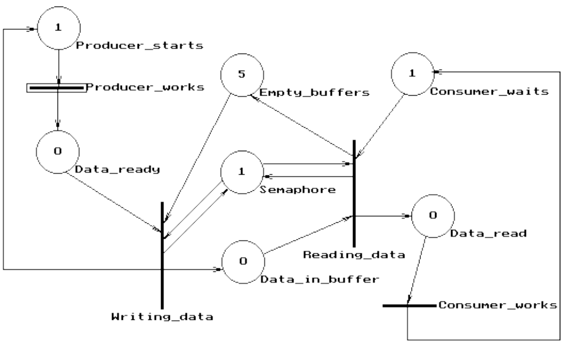


Fig. 3: An example of a Petri net (Source: Wikipedia)

4. Transformation of EMF Model into Petri Net

The idea of our method is that the input model has to be transformed into a description of the output as a Petri net. The input model contains components (like „House 1“ or „Kitchen and dining room“). Any component can contain other components (like „House“ contains „Kitchen and dining room“), or elements (like „Boiler room“ contains „Boiler“). Components have input and output ports, which are connected to each other.

The first rule states that components will be interpreted as Petri net places. For each component in the input we have to create a place with the same identification in the output. E.g. the component „Kitchen and dining room“ will be converted into the place „Kitchen and dining room“ in the output net.

The second rule says, that interconnections between components should be realized by Petri net transitions. E.g. the connection between the output port „temp1“ of „Hall and stairway“ and the input port „temp1“ of the „Scheduler“, should be implemented by the transition „temp1“ in the output net. The transition in the output model will have two arcs in two directions – from „Hall and stairway“ component into „Scheduler“, which we call „has“, and at the opposite direction, which we call „s“. The transition will have assigned two predicates, derived from the input description and attached ports:

```
has: output(temp1);
s: output(temp1);
```

The output names „has“ and „s“ are derived from abbreviated names of connected components (“has“ for „Hall and stairway“, and „s“ for „Scheduler“). The result should be accomplished with Petri nets of elements like sensors, actuators and so on.

5. Transformation of Petri Nets into PNML

To show a high-level implementation of such a transformation, we begin by decomposing the EMF format as that is our only input. For this purpose let us say that:

- Let $C = \{ C_i \}$ be a set of all **`<contains>`** elements in depth 1, such that they match the xPath expression: **`component/contains/contains`**.
- Let $D = \{ D_i \}$ be a set of all **`<dataflow>`** elements in depth 1, such that they match the xPath expression: **`component/contains/dataflow`**.

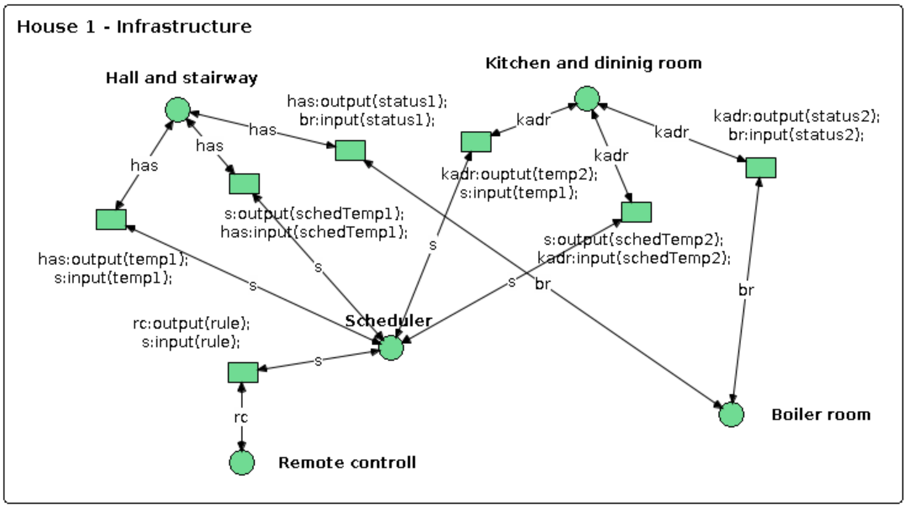


Fig. 4: Resulting infrastructure of the small house

Now the desired mapping from EMF to PNML can be described by the following algorithm. We suppose that we are able to extract components and dataflows from the model in EMF.

The Algorithm for transformation of EMF to PNML

Input: The EMF document the workflow description of the agent, and let $C = \{ C_i \}$ and $D = \{ D_i \}$ be two sets of all components, resp. dataflows contained in it.

Output: The PNML document containing the Petri net graph $G = (P,T,W)$, which is behaviorally equivalent to the input.

Process:

1. Set $P = \emptyset$, $T = \emptyset$, $W = \emptyset$.
2. For every C_i from C create a place P_i such that the attribute *id* of P_i is unique across the entire document and set $P = P \cup \{ P_i \}$.
3. For every D_j from D create a transition T_j such that the attribute *id* of T_j is unique across the entire document and set $T = T \cup \{ T_j \}$.
4. For every D_j from D create a pair of arcs A_1 and A_2, such that the attribute *id* of A_1 and A_2 is unique across the entire document, set $W = W \cup \{<A_1 , A_2>\}$ and:
   a. Attribute **`from`** of the element A_1 has the value of attribute *id* of the place P_i which was created from an element C_i referenced by the first expression in attribute ports.
   b. Attribute **`to`** of the element A_1 has the value of attribute *id* of the transition T_j which was created from dataflow D_j.
   c. Attribute **`from`** of element A_2 has the value of attribute *id* of the place P_i which was created from an element C_i referenced by the second expression in attribute ports of D_j.
   d. Attribute **`to`** of element A_2 has the value of attribute id of the transition T_j which was created from dataflow D_j.

Sketch of the proof:

Due to the step 2, all components of EMF model C will have appropriate place in the Petri net graph G. Due to the step 3, all dataflows in D will have appropriate transition in the Petri net graph G. In the step 4, all transitions created in the step 3 will be accomplished by input and output arcs according to the source dataflow. In such a way, when input places of the transition T_j contains sufficient marking, the transition can fire, and result is the transition of marking from all input places to all output places.

q.e.d.

The example:

Let us suppose the example of heating system from the Fig: 1. The set C contains (among others) two components: „Hall and stairway", and „Scheduler". So the set P will contain two places, e.g.: „Hall-and-stairway", and „Scheduler". The set D contains (among others) two dataflows: „temp1", and „schedTemp1". So the set T will contain two transitions, e.g.: „temp1", and „schedTemp1". The set W will contain arcs <Hall-and-stairway.temp1, Scheduler.temp1>, and <Scheduler.schedTemp1, Hall-and-stairway.schedTemp1>. It means, that the Petri net graph can fire transition $M \rightarrow_G M'$ such that changes the marking M into the marking M' according to the definition of this arc. So the signal of the workflow model is correctly simulated by the constructed Petri net.

Comments to XSLT Implementation of the Algorithm

Step 1 can be achieved with a simple pair of „**template**“ and „**apply-template**“ constructs, where „**template**“ may look like

```
<xsl:template match="contains">
  <place>
    <xsl:attribute name="id">
      <xsl:value-of select="generate-id()"/>
    </xsl:attribute>
    <name>
      <text>
         <xsl:value-of select="@name"/>
       </text>
    </name>
  </place>
</xsl:template>
```

It is important to note, that using the XSLT function „**generate-id()**“ for id generation will not be enough since the virtual machine relies on numerical-only ids, while „**generate-id()**“ generates an alphanumerical string. For the id we will use the following element:

```
<xsl:number format="0000000" level="any"/>
```

that returns the number of preceding elements (for the current element). The number length has to be chosen with respect to the potential number of unique elements in the Petri net.

Step 2: Can be achieved in a similar manner.

Step 3: The XSLT limitation becomes obvious with the step 3. Creating a pair of „**arc**“ elements would be a trivial problem, not given by conditions. To illustrate why this point is problematic, please let us consider a significantly simplified structure of the EMF input:

```
<house>
  <contains name=”knob”/>
  <contains name=”sensor”/>
  <dataflow ports=”sensor knob”/>
</house>
```

Now by the algorithm described above, in the first step, we would transform the elements „**knob**“ and „**sensor**“ into elements „**place**“ with unique ids. Step 2 is also feasible in the same manner. In step 3 however, we need to make a pair of arcs, that both specify, which nodes in PNML they connect. Since XSLT is a declarative programming language which means that we do not describe the control flow, but rather describe the logic of the computation. This also means that XSLT lacks any sort of data structures resembling a Key, Value map. This means, that in the time of

„**arc**“ elements creation, we do not know what PNML ids the elements (we cannot reference elements in the document that is being created) „**knob**“ and „**sensor**“ have and therefore cannot reference them correctly.

An argument could be made that if the „**place**“ elements would be created at the time the arcs are, this problem could be omitted, since we could create a „**xsl:variable**“ and store these ids temporarily. This would be a viable solution if every „**place**“ had only one connection to another place. To demonstrate please consider the following structure of the EMF input:

```
<house>
  < contains name=”knob”/>
  < contains name=”sensor”/>
  < contains name=”web-client”/>
  <dataflow ports=”sensor knob”/> (1)
  <dataflow ports=”sensor web-client”/> (2)
</house>
```

In this example, we would indeed correctly reference both „**sensor**“ and **„knob“** from transforming (1). The XSLT processor would then proceed to process (2) and that is where we would get to the same problem as before.

A workaround for this issue will be to run 2 consecutive transformations and split the algorithm into 2 parts. The first transformation will copy every existing element, but at the same time add **„place“** elements as per the description of the step 1.

The second transformation will use this enriched document and process steps 2 and 3. To demonstrate why this solves our problem, please consider the following EMF input:

```
<house>
  <contains name=”knob”/>
  <contains name=”sensor”/>
  <dataflow ports=”sensor knob”/>
</house>
```

2nd transformation input:

```
<house>
  <contains name=”knob”/>
  <place name=”knob” id=”1”/>
  <contains name=”sensor”/>
  <place name=”sensor” id=”2”/>
  <dataflow ports=”sensor knob”/>
</house>
```

Please note that the **„place“** elements have an additional attribute name. That is necessary so that when we are processing the **„dataflow“** element we can correctly match **„place“** and **„contains“**.

Creating the **„transition“** and **„arc“** elements is trivial now – we can retrieve the „**place@id**“ attributes by making a simple conditional search over all **„place“**

elements which have the attribute „`name`“ equal to the first/second part of the „`ports`“ attribute in the currently processed „`dataflow`“.

The specific implementation of the XSLT scripts is slightly more technically demanding. As an example, we should mention that the „`dataflow@ports`“ attribute does not use a unique name, like in the example, but rather an expression not completely un-like xPath. These expressions resemble xPath, but have slightly different syntax, which makes it impossible to use them for referencing without proper translation. For example, the following expression:

```
//@contains.0/@contains.1/@inputPorts.0
```

is referencing an „`inputPorts`“ node, which has two ancestors „`contains`“. The parent „`contains`“ is a second child node to the root „`contains`“ node.

```
<pnml xmlns="RefNet">
   <net id="netId1479076635752" type="RefNet">
      <place id="56">
         <name>
            <graphics>
               <offset x="4" y="-26"/>
            </graphics>
            <text>Hall and stairway</text>
         </name>
         <graphics>
            <position x="161" y="117"/>
            <dimension x="20" y="20"/>
            <fill color="rgb(112,219,147)"/>
            <line color="rgb(0,0,0)"/>
         </graphics>
      </place>
      <place id="57">
         <name>
            <graphics>
               <offset x="6" y="-29"/>
            </graphics>
            <text>Kitchen and dininig room</text>
         </name>
         <graphics>
            <position x="493" y="108"/>
            <dimension x="20" y="20"/>
            <fill color="rgb(112,219,147)"/>
            <line color="rgb(0,0,0)"/>
         </graphics>
      </place>
. . .
```

Fig. 5: Example of a PNML text

However, even if these were pure xPath, we could not use them for referencing since XSLT 2.0 (currently newest non-commercial version) does not support dynamic xPath resolution (without any extensions). Problems such as these however are solvable on a programming level and do not require a rework of the algorithm.

6. PNML and Petri Net correlation

As described in [2] PNML is a sufficiently developed language to describe Petri nets like these (containing places, transitions and arcs). Suppose, we have some Petri net, e.g. the infrastructure of our small house, consider Fig. 4. As we can see, places correspond with components of the input model, transitions correspond to connections, and names of arcs are generated during the transformation. The position in a space is supplemented artificially, it is not included in the source model.

This Petri net could be expressed in the PNML format [13], for the fragment of it, see Fig. 5.

7. Conclusions

In the foregoing text we have shown that any Petri net, which we received from the input description using the described transformation, we can engage in a full description of the system and along with him to interpret. So for the description of any system, we can describe particular agents using classical models. These models are then transformed into the corresponding Petri nets. The communication between agents will be described using so-called an infrastructure and a platform layer [11]. For the effective usage of such a method we have to create supporting tools. It will be the subject of our further research.

We described the basics of model transformation and execution-based methodology of distributed embedded control system development. Among the main methods it uses Petri Nets models transformations and target system prototype code generation. The development process starts with the classical model of the system specification.

This model of the system describes the functionality from users' or domain specialist's point of view. Using our methods, this model is further transformed to the multi-layered architecture based set of Reference Petri Nets. Each layer of the system is then translated to the specific target representation called Petri Nets Byte Code (PNBC), which is interpreted by the Petri Nets Virtual Machine (PNVM), which is a part of the Petri Nets Operating System (PNOS), which is installed on all nodes of the system. Targeted system is dynamically reconfigurable by the possibility of PNBC net templates and instances replacement with its new versions. After the replacement, PNVM interpretation engine starts to perform a new version of partial functionality of the system.

Acknowledgments

This research has been partially supported by the Avast Foundation (2017), and also by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), by BUT FIT grant FIT-11-1, and by the Ministry of Education, Youth and Sports under the contract MSM 0021630528.

References

1. Biermann, E. - et al.: Graphical Definition of In-place Transformations in the Eclipse Modeling Framework. In: *International Conference on Model Driven Engineering Languages and Systems*, pp. 425-439. Springer Berlin Heidelberg 2006.
2. van Hee, K.M. – Sidorova, N. - van der Werf, J.M.: Business Process Modeling Using Petri Nets. In: Jensen, K. - van der Aalst, W.M.P. – Balbo, G. – Koutny, M. - Wolf K. (eds): Transactions on Petri Nets and Other Models of Concurrency VII. Lecture Notes in Computer Science, vol. 7480. Springer, Berlin, Heidelberg 2013.
3. Hillah, L.M. - Kindler, E. – Kordon, F. – Petrucci, L. – Trèves, N.: The Petri Net Markup Language and ISO/IEC 15909-2. CPN Workshop 2009.
4. Miori, V. – Tarrini, L. – Manca, M. – Tolomei, G.: An open standard solution for domotic interoperability. IEEE Transactions on Consumer Electronics, Year: 2006, Volume: 52, Issue: 1, pp: 97 - 103, DOI: 10.1109/TCE.2006.1605032.
5. OMG (February 2009). OMG Unified Modeling Language (OMG UML), Superstructure Version 2.2. URL: http://www.omg.org/spec/UML/2.2/Superstructure/PDF (2009).
6. Peterson, J.: Petri Nets. ACM Computing Surveys 9 (3), doi:10.1145/356698.356702, (1977) 223–252.
7. Petri, C.A.: Kommunikation mit Automaten. Ph. D. Thesis. University of Bonn (1962).
8. Petri, C.A., Reisig, W.: Petri net. URL: http://www.scholarpedia.org/ (2008), 3(4):6477. Retrieved 2008-07-13.
9. Richta, T. - Janoušek, V.: 2013. Operating System for Petri Nets-Specified Reconfigurable Embedded Systems. In: *Proceedings of Computer Aided Systems Theory - EUROCAST 2013*, published as LNCS 8111, pp.444-451. ISBN 978-3-642-53855-1. Berlin Heidelberg: Springer Verlag, 2013.
10. Richta, T. - Janoušek, V. – Kočí, R.: Code Generation For Petri Nets-Specified Reconfigurable Distributed Control Systems. In: *Proceedings of 15th International Conference on Mechatronics - Mechatronika 2012*, pp.263-269. ISBN 978-80-01-04985-3. Prague, 2012.
11. Richta, T. - Janoušek, V. – Kočí, R.: Petri Nets-Based Development of Dynamically Reconfigurable Embedded Systems. In: *Proceedings of PNSE'13 - CEUR Workshop Proceedings*, Vol. 2013, Issue 989, str.203-217. ISSN 1613-0073. Hamburk, 2013.
12. Steinberg, D, - Budinsky, F. - Merks, E. – Paternostro, M.: EMF: Eclipse Modeling Framework. Pearson Education, 2008.
13. Weber, M. – Kindler, E.: The Petri Net Markup Language. In: *Petri Net Technology for Communication-Based Systems – Advances in Petri Nets*, LNCS volume 2472, pp.124-144, 2003.
14. W3C Standards - XSLT Current Status [Retrieved 2017-03-21], URL: https://www.w3.org/standards/techs/xslt#w3c_all, 2017.

Analysing Musical Pieces Using harmony-analyser.org Tools

Ladislav Maršík

Dept. of Software Engineering, Faculty of Mathematics and Physics
Charles University, Malostranské nám. 25, 118 00 Prague 1, Czech Republic
marsik@ksi.mff.cuni.cz

Abstract. The tools provided under *harmony-analyser.org* are capable of recognizing harmonies, extracting the high-level harmony features, and plotting the harmony structure of the audio. They focus on the classical tonal analysis, as well as the distances between the harmonies to allow for the creation of novel descriptors. In the light of the recent expansion of the music retrieval techniques, the concepts of chord distances or chroma vector distances were still not studied to the full extent. With the presented tools we aim to provide an easy-to-use system for anyone interested in extracting these features, as well as an open-source framework written in Java for the developers interested in researching the concepts further. In this short paper, we offer the walk-trough of *harmony-analyser.org* tools with the manual for the correct usage. We also summarize the results achieved using our system and we set the focus for the next development and research.

Keywords: harmony-analyser.org, tonal analysis, chord distance, chroma vector distance, music information retrieval

1 Introduction

The focus in Music Information Retrieval (MIR) is recently shifting to the large-scale approaches and techniques for a fingerprint extraction in the way that the most relevant audio features are retained [2]. The research teams are using fingerprints based on the spectrogram analysis [16], music theory [6], and most recently also experimenting with deep learning techniques to learn and distinguish musically relevant features [14].

In the search for the best features and fingerprints, it becomes increasingly difficult to know what features are already available. There are many proposed methods of extraction, as we can clearly see on benchmarking challenges such as MIREX[1] with over 15 distinct tasks, each requiring a different set of audio features, and the features changing every year since the first benchmarking in 2005.

[1] `http://www.music-ir.org/mirex/wiki/MIREX_HOME`

On the other hand, MIR is an interdisciplinary field and there are not many institutions worldwide having their own MIR team or laboratory. Therefore it can be challenging, especially for the young researchers, to join the common effort and be a part of the MIR project, unless a similar project is hosted by the researcher's academic institution. To fulfil the need of onboarding the new researchers, popularizing the MIR field, giving an overview of the common techniques, and facilitating an open-source system, we have started the *harmony-analyser.org* project in 2016 [9].

Analysing harmonies is the main, but not the only aim of the project. The analysis output (harmony features) can be used for further retrieval easily, e.g. by employing Dynamic Time Warping (DTW) techniques [13]. We chose to focus on harmony to honour the fact that a musical piece usually contains multiple instruments played simultaneously, and the resulting harmony is one of the main features used for retrieval [6]. But the project is open for analysing melodies, rhythm, or beat tracking in the future, as well as using the machine learning approaches instead of the traditional feature extraction.

We continue by introducing the reader to the concepts and related work in Section 2. The step-by-step manual for the tools with screenshots is presented in Section 3. The first results obtained by our techniques are summarized in Section 4 and our future work is discussed in Section 5.

2 Harmony Features and Related Work

Chroma features is a common name for a series of 12-dimensional vectors of floating-point numbers, capturing the presence of each tone in a short music moment. They became popular after the works of Fujishima [5] and Bartsch and Wakefield [1]. Obtained directly from the Discrete-Time Fourier Transform output by grouping frequencies that belong together in one frequency bin, the resulting chroma vector has the form:

$$< c_A, c_{A\#}, c_B, c_C, c_{C\#}, c_D, c_{D\#}, c_E, c_F, c_{F\#}, c_G, c_{G\#} >$$

where $c_A \in \mathbb{R}$ represents the presence of the A tone, $c_{A\#} \in \mathbb{R}$ represents the presence of $A\#$ tone, etc. The value distribution of $c_A, c_{A\#}, \ldots$ depends on the algorithm used, but it is a common practice to normalize to $[0, 1]$ interval, where the value represents the loudness of the frequency bin. We refer the reader to Bartsch and Wakefield [1] for a detailed definition. One of the motivations for our work is, that chroma vectors have not yet been studied in terms of distances, even though the distances in between the chords have long been proposed by the works in music cognition [7].

Chord progression (a sequence of chord labels) is a familiar concept for musicians, who often use it to play together in an unrehearsed situation. The idea of using chord progression itself as a fingerprint for large-scale music retrieval was proposed by Khadkevich and Omologo [6], improving the state-of-the-art cover song identification results in 2013. The progression can be represented as a sequence of strings (*C, F6, Gmaj7, ...*), or boolean vectors similar to chroma vectors.

Chord distance is a concept based on the acknowledged music cognition findings: the listeners perceive the differences in chords in a way that can be predicted by a formal tonal harmony model. Fred Lerdahl's Tonal Pitch Space (TPS) model [7] was proposed and backed up by the empirical studies. This concept was further studied by several MIR authors [4] [12] [15], combining the cognitive and computational chord distances. A thorough review of the available chord distances was assembled by Rocher et al. [15]. Notably, the TPS distance performed the best in the studies for the chord estimation or cover song identification tasks [12] [15].

3 Usage of Harmony Analyser Tools

In *harmony-analyser.org* project, we provide GUI tools published as executable JAR archives, to allow for a custom harmony analysis of WAV or MIDI input. The tools itself are using the *JHarmonyAnalyser* Java library, which we describe in details in the more technical report [9]. To achieve a high variety of analysis, we also incorporated GPL-licensed Vamp plugins[2] to the GUI tools. The advanced users can customize their analysis by downloading additional plugins or creating their own.

In this section we focus on a simple use case of running the tools to get a simple analysis of the MIDI keyboard input and WAV files. We also describe the differences from the other systems and possible usages for the research along the way.

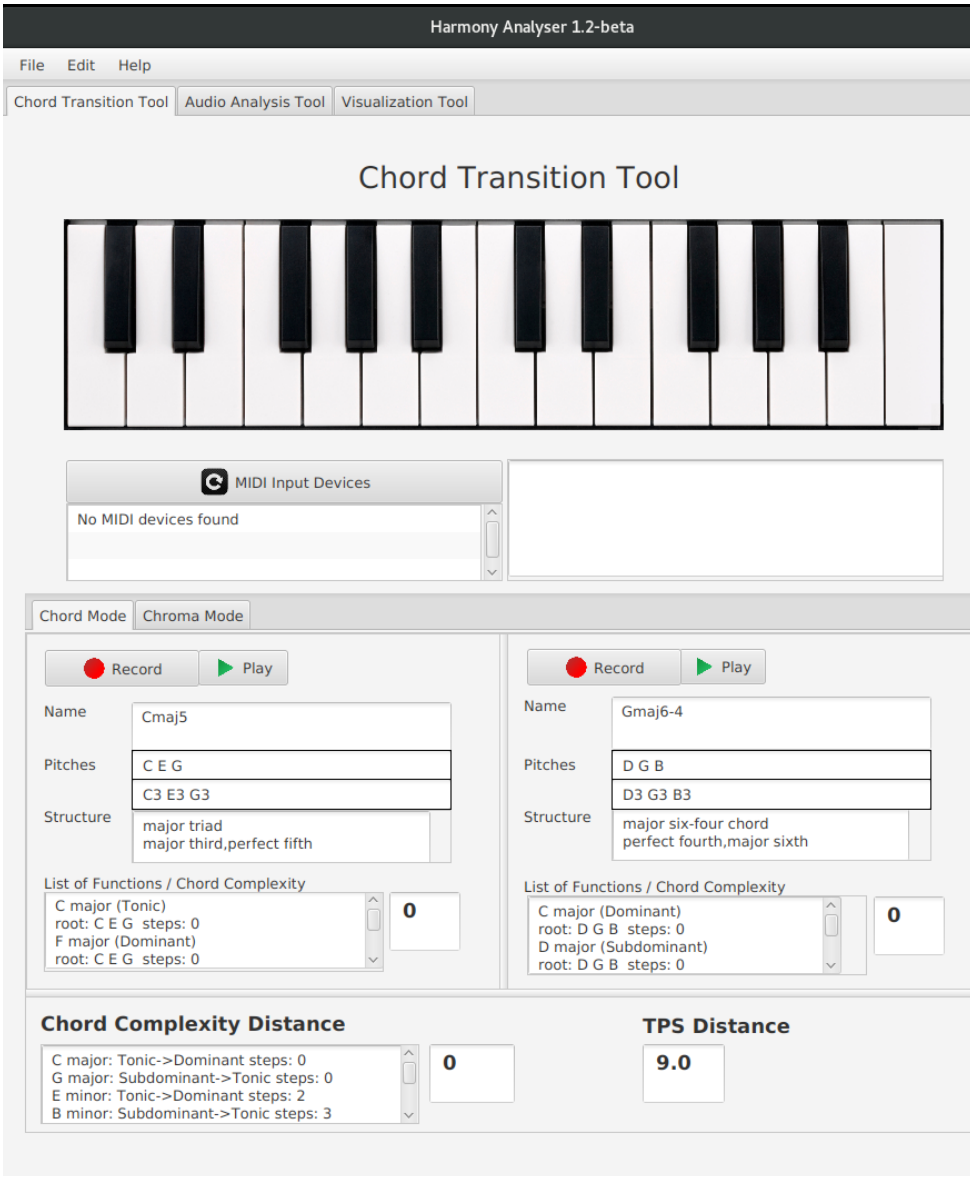
3.1 Chord Transition Tool

When the application starts, the default tool selected is the *Chord Transition Tool* (see Figure 1). The user can either use the MIDI keyboard plugged in via the USB port, or use a text input field, to specify two chords. The added value compared to other common MIDI software is a list of functions and chord distances, based on the tonal analysis (described in more details in [10]). The fact that the chord can have multiple functions in music is commonly accounted for in the works on musicology, but less frequently in the MIR works. This is one of the many examples of a gap between MIR and musicology, which should be addressed, as pointed out by Lewis [8]. Chord Transition Tool shows the chord and all of its tonal functions, and the user can observe various chord distances (Chord Complexity Distance [10], or TPS Distance [7]) as seen on Figure 1, which gives him a good overview for developing advanced tonal features.

3.2 Visualization Tool

After the user is familiar with chordal analysis described in the previous section, the next step is to observe the chords, chord distances, or chroma vector distances

[2] `http://vamp-plugins.org`



Chord Transition Tool

No MIDI devices found

Name: Cmaj5
Pitches: C E G
C3 E3 G3
Structure: major triad
major third,perfect fifth

List of Functions / Chord Complexity
C major (Tonic)
root: C E G steps: 0
F major (Dominant)
root: C E G steps: 0
0

Name: Gmaj6-4
Pitches: D G B
D3 G3 B3
Structure: major six-four chord
perfect fourth,major sixth

List of Functions / Chord Complexity
C major (Dominant)
root: D G B steps: 0
D major (Subdominant)
root: D G B steps: 0
0

Chord Complexity Distance
C major: Tonic->Dominant steps: 0
G major: Subdominant->Tonic steps: 0
E minor: Tonic->Dominant steps: 2
B minor: Subdominant->Tonic steps: 3
0

TPS Distance
9.0

Fig. 1. Chord Transition Tool: capturing the MIDI input and outputting the chord labels, functions and the chord distances. *C major* and *G major* chords are analysed.

extracted from the real audio. We offer the Visualization Tool (see Figure 2) to visually understand how the labels and distances can help analysing a musical piece. In the musical piece analysis on Figure 2 (Hallelujah by Bastian Baker) we have time in seconds on the x axis, and chord distance values of *each pair of the subsequent chords* on the y axis. This is one of the song fingerprints that we experimentally studied. In the given analysis, the chord distance time series represents a typical curvature of the harmony movement in the piece. The local peaks around 30th or 80th second represent the transition between *Ami* and *F* chords. The peaks after the 150th second represent the same transition with the singer performing vocal ornaments in the last verse, yielding a higher (more complex) chord distance value, since the voice is accounted for in the chord estimation.

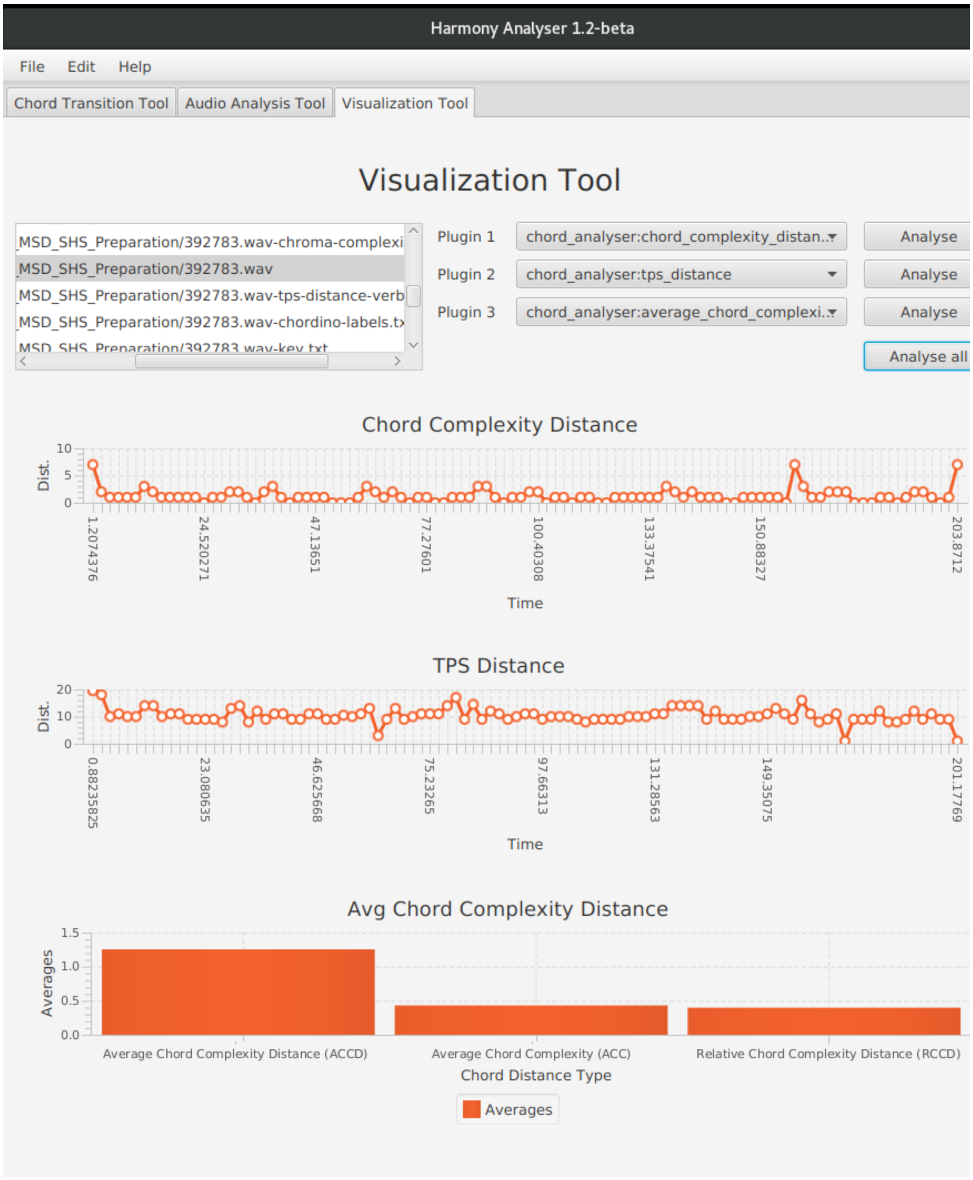
The same chart visualization can be shown for each plugin that extracts the values in the form of a time series (e.g. chroma vector distances), or labels with a timestamp (chord or key detection). Some plugins will output column charts, such as Average Chord Complexity Distance [11] on Figure 2. We have shown how these averaged features improve the music genre detection in one of our previous studies [10].

3.3 Audio Analysis Tool

The last step of the analysis after understanding the harmony features thoroughly, is to apply the chosen analysis on a folder with WAV files. This can be achieved by the Audio Analysis Tool (Figure 3). The plugins are categorized in the plugin groups (*Vamp plugins, Chord analyser, Chroma analyser*) and the details and parameters of the selected plugin are shown. After hitting the *Analyse* button, the tool creates text files with the analysis results in the selected folder. These can be used as an input for another analysis plugin, or an input for a retrieval technique. There is also an additional *Post Processing* tab that serves various purposes, such as applying a smoothing filter to a time series. The additional tabs can also be helpful for importing or exporting other file types, so that the application can be used for various projects. As an example, The Million Song Dataset from Bertin-Mahieux et al. [3] uses *HDF5* files, and by providing a conversion to text files the dataset can be used easily with Audio Analysis Tool.

4 Summary of Results

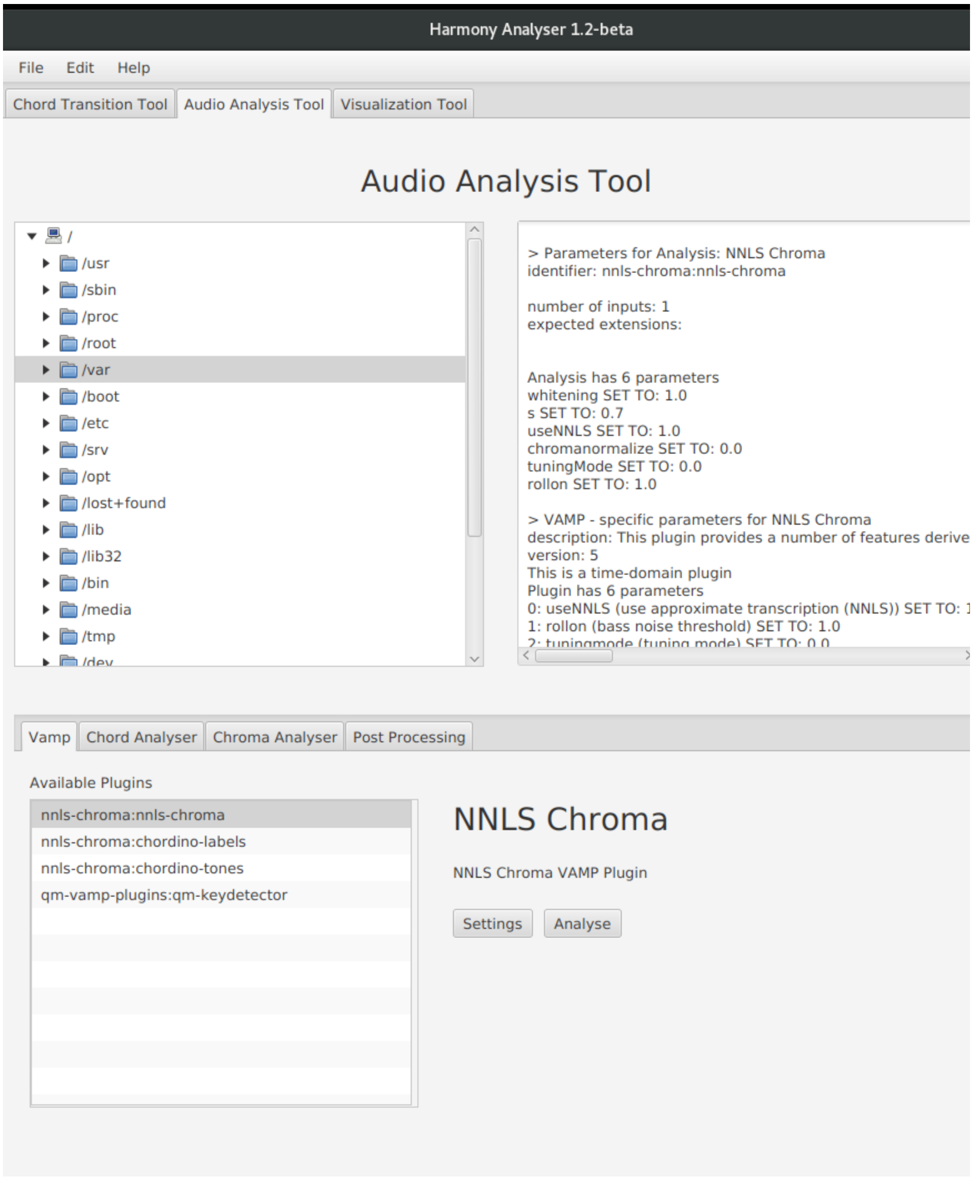
The tools from *harmony-analyser.org* have already been tested on various MIR tasks. We have gathered an average chord complexity distance and used this average for the genre detection, as one of the features for the neural network method. The usage of the feature yielded to 4% precision improvement for the dataset of 100 musical pieces [10].


Harmony Analyser 1.2-beta
File
Edit
Help
Chord Transition Tool
Audio Analysis Tool
Visualization Tool
Visualization Tool
Analyse
Analyse
Analyse
Analyse all
Chord Complexity Distance
Dist.
Time
TPS Distance
Avg Chord Complexity Distance
Averages
Average Chord Complexity Distance (ACCD)
Average Chord Complexity (ACC)
Relative Chord Complexity Distance (RCCD)
Chord Distance Type
Averages

MSD_SHS_Preparation/392783.wav-chroma-complexi
MSD_SHS_Preparation/392783.wav
MSD_SHS_Preparation/392783.wav-tps-distance-verb
MSD_SHS_Preparation/392783.wav-chordino-labels.tx
MSD_SHS_Preparation/392783.wav-key.txt

Plugin 1 chord_analyser:chord_complexity_distan..
Plugin 2 chord_analyser:tps_distance
Plugin 3 chord_analyser:average_chord_complexi..

Fig. 2. Visualization Tool: Analysis of Hallelujah by Bastian Baker is shown, containing results for Chord Complexity Distance, TPS Distance, and three types of averages for Chord Complexity Distance [11].



```
> Parameters for Analysis: NNLS Chroma
identifier: nnls-chroma:nnls-chroma

number of inputs: 1
expected extensions:


Analysis has 6 parameters
whitening SET TO: 1.0
s SET TO: 0.7
useNNLS SET TO: 1.0
chromanormalize SET TO: 0.0
tuningMode SET TO: 0.0
rollon SET TO: 1.0

> VAMP - specific parameters for NNLS Chroma
description: This plugin provides a number of features derive
version: 5
This is a time-domain plugin
Plugin has 6 parameters
0: useNNLS (use approximate transcription (NNLS)) SET TO: 1
1: rollon (bass noise threshold) SET TO: 1.0
2: tuningmode (tuning mode) SET TO: 0.0
```

Fig. 3. Audio Analysis Tool: selecting a folder with WAV files and choosing a desired plugin for analysis.

The chord distance time series were tested on both covers80 dataset[3] and a subset of SecondHandSongs dataset[4] (999 songs), on a cover song identification task using DTW method [12]. The results show that TPS distance have outperformed Chord Complexity Distance in the MAP (Mean arithmetic of Average Precision) score. Overall, the usage of a chord distance time series means a loss in the MAP score compared to more low-level features: from 0.482 (full chroma features) to 0.198 (TPS distance) for covers80 dataset, but it comes with a more than a two thousand times faster performance (56s versus 25ms execution time for DTW matrix calculation of 80 songs).

The chroma vector distances were tested on the same datasets and task. The results were comparable to the results of a TPS chord distance (0.174 MAP score), which is promising for a first feature of this type.

These experiments show that the chord or chroma vector distance features do not provide enough information on their own for the retrieval, but if used properly in the combination with more low-level features, they can improve the performance.

5 Conclusion and Future Work

The *harmony-analyser.org* tools can be used for a musical piece analysis, feature extraction from audio files, or as a basis for further research and retrieval. They contain a variety of plugins for analysis, giving a thorough overview of what harmony features are currently available. The tools are also extensible in the way that new plugins can be downloaded or developed. We provided an overview of the usage of the main tools, and the summary of the achieved results, showing the ways to enhance the algorithms for MIR tasks.

Our latest ideas were to utilize the concepts of chord and chroma vector distances differently. Rather than a stand-alone time series, we will be experimenting with using the distances in DTW calculation (comparison of two vectors done by the chord or chroma vector distance instead of the Euclidean distance). We also plan to include more types of chord distances in our tools to get a thorough comparison. Last but not least, we will continue to present the tools in the open-source community, to get more developers for the project, with the overall aim to make *harmony-analyser.org* an all-in-one music retrieval system.

Acknowledgments. The study was supported by the Charles University in Prague, project GA UK No. 1580317.

Bibliography

1. Bartsch, M.A., Wakefield, G.H.: To Catch a Chorus: Using Chroma-Based Representations for Audio Thumbnailing. In: *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics.* WASPAA 2001 (2001)

[3] `https://labrosa.ee.columbia.edu/projects/coversongs/covers80`
[4] `https://labrosa.ee.columbia.edu/millionsong/secondhand`

2. Bertin-Mahieux, T., Ellis, D.P.W.: Large-Scale Cover Song Recognition Using Hashed Chroma Landmarks. In: *IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics*. WASPAA 2011, IEEE (2011)
3. Bertin-Mahieux, T., Ellis, D.P., Whitman, B., Lamere, P.: The Million Song Dataset. In: *Proceedings of the 12th International Society for Music Information Retrieval Conference*. ISMIR 2011 (2011)
4. De Haas, W.B., Veltkamp, R., Wiering, F.: Tonal Pitch Step Distance: A Similarity Measure for Chord Progressions. In: *Proceedings of the 9th International Conference on Music Information Retrieval*. ISMIR 2008 (2008)
5. Fujishima, T.: Realtime Chord Recognition of Musical Sound: A System Using Common Lisp Music. In: *Proceedings of the International Computer Music Conference*. ICMC 1999 (1999)
6. Khadkevich, M., Omologo, M.: Large-Scale Cover Song Identification Using Chord Profiles. In: *Proceedings of the 14th International Society for Music Information Retrieval Conference*. ISMIR 2013 (2013)
7. Lerdahl, F.: Tonal Pitch Space. Oxford University Press, Oxford (2001)
8. Lewis, R.J., Fields, B., Crawford, T.: Addressing the Music Information Needs of Musicologists. In: *Proceedings of the 16th International Society for Music Information Retrieval Conference*. ISMIR 2015 (2015)
9. Marsik, L.: harmony-analyser.org - Java Library and Tools for Chordal Analysis. In: *Proceedings of 2016 Joint WOCMAT-IRCAM Forum Conference*. WOCMAT 2016, Kainan University, Taiwan (2016)
10. Marsik, L., Pokorny, J., Ilcik, M.: Improving Music Classification Using Harmonic Complexity. In: *Procedings of the 14th conference Information Technologies - Applications and Theory (ITAT 2014)*. Ústav informatiky AV ČR (2014)
11. Marsik, L., Pokorny, J., Ilcik, M.: Towards a Harmonic Complexity of Musical Pieces. In: *Proceedings of the 14th Annual International Workshop on Databases, Texts, Specifications and Objects (DATESO '14). CEUR Workshop Proceedings*, vol. 1139. CEUR-WS.org (2014)
12. Marsik, L., Rusek, M., Slaninova, K., Martinovic, J., Pokorny, J.: Evaluation of Chord and Chroma Features and Dynamic Time Warping Scores on Cover Song Identification Task. In: *Proceedings of the 16th International Conference on Computer Information Systems and Industrial Management Applications*. CISIM 2017, Springer (2017)
13. Müller, M.: Information Retrieval for Music and Motion. Springer Berlin Heidelberg (2007)
14. Pons, J., Lidy, T., Serra, X.: Experimenting with Musically Motivated Convolutional Neural Networks. In: *14th International Workshop on Content-based Multimedia Indexing*. CBMI 2016, IEEE (2016)
15. Rocher, T., Robine, M., Hanna, P., Desainte-Catherine, M.: A Survey of Chord Distances With Comparison For Chord Analysis. In: *Proceedings of the International Computer Music Conference*. ICMC 2010 (2010)
16. Wang, A.L.: An Industrial-Strength Audio Search Algorithm. In: *Proceedings of the 4th International Society for Music Information Retrieval Conference*. ISMIR 2003 (2003)

Author Index