Variability Identification by Selective Targeting of Significant Nodes

Anilloy Frank
Institute of Technical Informatics,
Technische Universitt
Inffeldgasse 16, 8010 Graz, Austria
Email: anilloy.frank@student.tugraz.at

Eugen Brenner
Institute of Technical Informatics,
Technische Universitt
Inffeldgasse 16, 8010 Graz, Austria
Email: brenner@tugraz.at

***Abstract*—The automotive industry is characterized by numerous product variants, often driven by embedded software. With ever increasing complexity of embedded software, the electrical/electronic models in automotive applications are getting enormously unmanageable. Significant concepts for modeling and management of variability in the software architecture are under development. Models are hugely hierarchical in nature with numerous composite components deeply embedded within projects comprising of Simulink models, implementations in legacy C, and other formats. Hence, it is often necessary to define a mechanism to identify reusable components from these that are embedded deep within. The proposed approach is selectively targeting the component-feature model (CF) instead of an inclusive search to improve the identification. We explore the components and their features from a predefined component node list and the features node vector respectively. It addresses the issues to identify commonality in identification, specification and realization of variants within a product development. Since the approach does not depend on the depth of the components or on its order, it serves well with all the scenarios, thereby exhibiting a generic nature. The results obtained are faster and more accurate compared to other methods.**

***Keywords*-Design Tools; Embedded Systems; Feature Extraction; Software Reusability; Variability Management.**

I. Introduction

Embedded systems are microcontroller-based systems built into technical equipment mainly designed for a dedicated purpose, where communication with the outside world occurs via sensors and actuators [1]. Although this definition implies that embedded systems are used as isolated units, there is also a trend to construct distributed pervasive systems by connecting several embedded devices, as noted by Tanenbaum and van Steen [2].

The current development trend in automotive software is to map software components on networked Electronic Control Units (ECU), which includes the shift from an ECU based approach to a function based approach. Also, according to data presented by Ebert and Jones, up to 70 electronic units are used in a car containing embedded software consisting of more than 100 million lines of object code, which is mainly responsible for the value creation of the car.

Variants of embedded software functions are vital in customizing for different regions (Europe, Asia, etc.), to meet regulations of the respective regions. Also different sensors / actuators, different device drivers, and distribution of functionality on different ECUs necessitate variants. Managing variability involves extremely complex and challenging tasks, which must be supported by effective methods, techniques, and tools [3].

Ebert and Jones present recent data about embedded software in [4], stating that the volume of embedded software is increasing between 10 and 20 percent per year as a consequence of the increasing automation of devices and their application in real world scenarios.

The proposed strategy is to introduce a variability identification layer. It intends to facilitate a reusable software solution. We start by analyzing the textual representation of the model structure. Based on this we form a concept to extract an element list to facilitate the identification of variability. Both implementation and evaluation of the proposed strategy is based on a technically advanced adaptation of a formal mathematical model, which is beyond the scope of this paper.

II. Software reuse

In the 1960s, reuse of software started with subroutines, followed by modules in the 1970s and objects in the 1980s. About 1990 components appeared, followed by services at about 2000. Currently, Software Product Lines (SPL) are state of the art in the reuse of software.

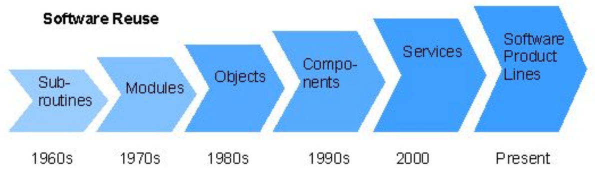


Figure 1. Software reuse history.

Figure 1 shows a short history of the usage of reuse in software development. The key idea of Product Lines is very old; it is based on Henry Ford's mass customization to provide a effective way for cheap individual cars. Today, many different approaches exist to the implementation of Software Product Lines.

A SPL is a set of software-intensive systems that share a common set of features for satisfying a particular market segment's needs. SPL can reduce development costs, shorten time-to-market, and improve product quality by reusing core assets for project-specific customizations [3][5].

Despite of all the hype, there is a lack of an overall reasoning about variability management.

The SPL approach promotes the generation of specific products from a set of core assets, domains in which products have well defined communalities and variation points[6].

Although variability management is recognized as an important issue for the success of SPLs, there are not many solutions available [7]. However, there are currently no commonly accepted approaches that deal with variability holistically at architectural level [8].

III. Variability management

One of the fundamental activity in Software Product Line Engineering (SPLE) is Variability management (VM). Throughout the SPL life cycle, VM explicitly represents variations of software artifacts, managing dependencies among variants and supporting their instantiations [3].

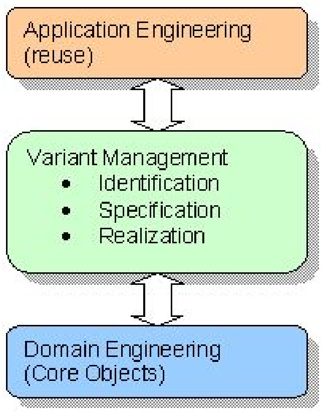


Figure 2. Variability management in product lines.

To enable reuse on a large scale, SPLE identifies and manages commonalities and variations across a set of system artifacts such as requirements, architectures, code components, and test cases. As seen in the Product Line Hall of Fame [9], many companies have adopted this development approach.

SPLE as depicted in Figure 2 can be categorized into domain engineering and application engineering [10][11]. Domain engineering involves design, analysis and implementation of core objects, whereas application engineering is reusing these objects for product development.

Activities on the variant management process involves variability identification, variability specification and variability realization [12].

- The Variability Identification Process will incorporate feature extraction and feature modeling.
- The Variability Specification Process is to derive a pattern.
- The Variability Realization Process is a mechanism to allow variability.

IV. Software Architecture

Figure 3 depicts a layered software architecture that is considered in the proposed architecture. It shows a comparison of distributed systems and platform with the proposed layered architecture and the feasibility of mapping the corresponding artifacts and responsibilities for each layer.

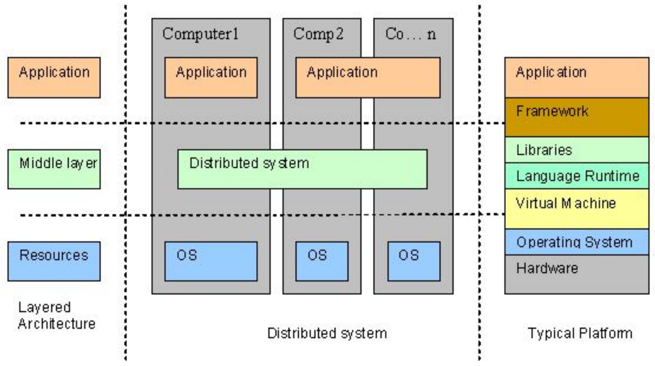


Figure 3. Comparison of architecture, system, and platform.

The definition of software architecture given in the ISO/IEC 42010 IEEE Std 1471-2000: "The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution [13]."

In the middle illustrates a distributed system. Tanenbaum and van Steen define distributed systems as "A distributed system is a collection of independent computers that appears to its users as a single coherent system [2]."

Similarly to the right side is depicted a typical platform as specified by Atkinson and Kühner in Model Driven Architectures (MDA) [14].

V. Specification of the Cases

To enable identification of variability for software components in a distributed system within the automotive domain [15][16], we enlist the specifications below:

- *Specification of components by compatibility*
  The product is tested using software functions of a certain variant and version. These products may exhibit compatibility issues between functional blocks, whilst

using later version of the function may fail to perform as expected.

- *Extract, identify, and specify features*
  To enable parallel development, it is necessary to be able to extract features, and to identify and specify the functional blocks in the repository based on architecture and functionality.
- *Usability and prevention of inconsistencies*
  A process that tracks usability and prevents inconsistencies due to deprecate variants and versions in the repository is required.
- *Testing mechanism for validations*
  A testing mechanism for validations in order to maintain high quality for components and its variants has to be established.
- *Mechanism for simplified assistance*
  The developer has to be assisted by a process to intelligently determine whether a functional block or its variant should exist in the data backbone to avoid redesign of existing functions, thereby improving productivity.

VI. Proposed approach for Variability Identification

Models confirming to numerous tools like ESCAPE®, EAST-ADL®, UML® tools, SysML® specifications and AUTOSAR® were considered. Although this concept is not limited to automotive domain alone.

A. Project analysis

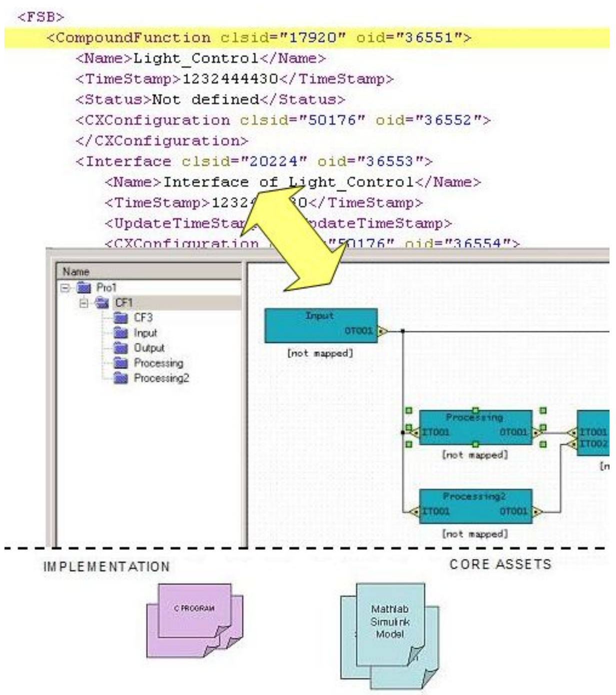
An analysis of the models exhibits a common architecture. Figure 4 depicts the textual representation that underlies several graphical model. The textual representation usually is given in XML, which strictly validates to a schema. A heterogeneous modeling environment may consist of numerous design tools, each with its own unique schema, to offer integrity and avoid inconsistencies. Developed projects have to be strictly validated to the schemas of these tools.

A closure examination of the nodes in the textual representation of models depicted in Figure 5 reveals some interesting information. The nodes outlined in rectangles provide important information regarding the identity, specification, physical attributes, etc. of a component, but are insignificant from the perspective of variant.

B. Concept and approach

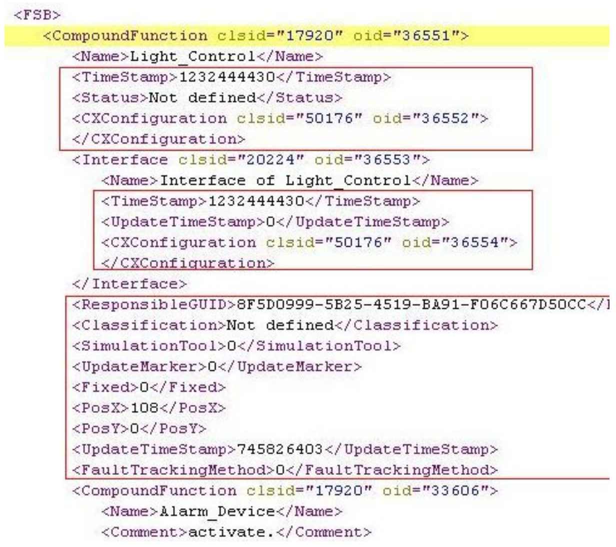
The basic concept to identify variability is depicted in Figure 6.

The left side is a set of projects that have software components hierarchically embedded. These projects validate to the corresponding schemas. The middle layer is an identification layer with three functional blocks. A set of component lists is derived from the node list in the schema. Similarly a feature vector is derived from it that corresponds to components.



```
<FSB>
    <CompoundFunction clsid="17920" oid="36551">
        <Name>Light_Control</Name>
        <TimeStamp>1232444430</TimeStamp>
        <Status>Not defined</Status>
        <CXConfiguration clsid="50176" oid="36552">
        </CXConfiguration>
        <Interface clsid="20224" oid="36553">
            <Name>Interface of Light_Control</Name>
            <TimeStamp>1232[illegible]0</TimeStamp>
            <UpdateTimeSta[illegible]dateTimeStamp>
            <CXConfiguration [illegible]"50176" oid="36554">
```

Figure 4. Mapping textual and graphical representations.



```
<FSB>
    <CompoundFunction clsid="17920" oid="36551">
        <Name>Light_Control</Name>
        <TimeStamp>1232444430</TimeStamp>
        <Status>Not defined</Status>
        <CXConfiguration clsid="50176" oid="36552">
        </CXConfiguration>
        <Interface clsid="20224" oid="36553">
            <Name>Interface of Light_Control</Name>
            <TimeStamp>1232444430</TimeStamp>
            <UpdateTimeStamp>0</UpdateTimeStamp>
            <CXConfiguration clsid="50176" oid="36554">
            </CXConfiguration>
        </Interface>
        <ResponsibleGUID>8F5D0999-5B25-4519-BA91-F06C667D50CC</I
        <Classification>Not defined</Classification>
        <SimulationTool>0</SimulationTool>
        <UpdateMarker>0</UpdateMarker>
        <Fixed>0</Fixed>
        <PosX>108</PosX>
        <PosY>0</PosY>
        <UpdateTimeStamp>745826403</UpdateTimeStamp>
        <FaultTrackingMethod>0</FaultTrackingMethod>
        <CompoundFunction clsid="17920" oid="33606">
            <Name>Alarm_Device</Name>
            <Comment>activate.</Comment>
```

Figure 5. XML Nodes that are not significant for variability.

The second block is a customized parser that generates a relevant lexicon from the set of software components within a project. The third block is a set of rules (viz., mandatory, optional, exclude) to govern the identification of variability.

The basic concept can be extended to obtain a working model for the identification of variants. The work flow is depicted in Figure 7. The top layer here represents the domain or core assets. The middle layer is a semi-automatic identification layer for variants. A component list and a

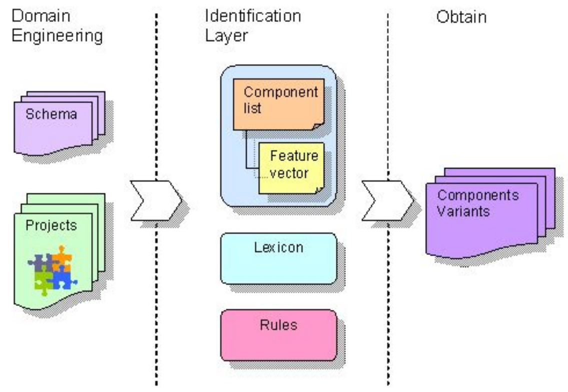


Figure 6. Basic Concept.

feature vector is derived manually from the schema of the project; a collection of elements that represent components and their descriptive features that significantly contribute to the identification of the component's variant.

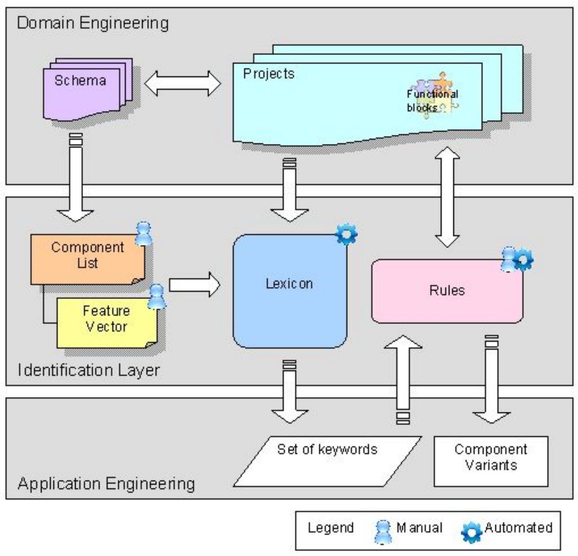


Figure 7. Work flow of the identification process.

The workflow can be further extended to adapt a heterogeneous environment which consist of projects developed using several modeling and simulation tools. The identification layer is separated into two parts. Numerous component lists and feature vectors can be derived for each distinct schema as depicted in Figure 8, whereas a common lexicon and common rules govern the identification process.

C. Evaluation

A prototype of the architecture presented here has been implemented. These case studies targeted the design of model-based software components firstly in an industrial use

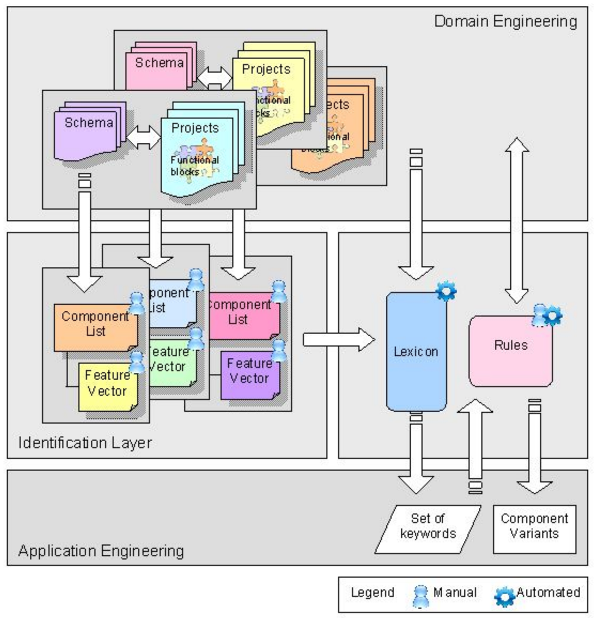
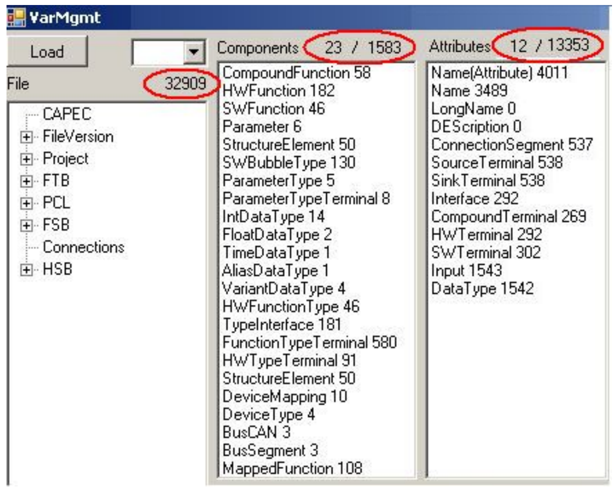


Figure 8. Work flow of the identification process for heterogeneous systems.

case where the project model was developed using the design tool ESCAPE® [17], and secondly in a case study targeting the execution of specific paradigms based on the naming convention of AUTOSAR® [18].

The specific project data set depicted in Figure 9, which was used to verify the implementation, consisted of a total of 32909 elements. Of these elements a total of 1583 elements signify components, these were categorized into 23 categories when enlisted in the component list. A total of 13353 elements signified features that were assigned into 12 categories.


VarMgmt

Load

File 32909

CAPEC
FileVersion
Project
FTB
PCL
FSB
Connections
HSB

Components 23 / 1583

CompoundFunction 58
HWFunction 182
SWFunction 46
Parameter 6
StructureElement 50
SWBubbleType 130
ParameterType 5
ParameterTypeTerminal 8
IntDataType 14
FloatDataType 2
TimeDataType 1
AliasDataType 1
VariantDataType 4
HWFunctionType 46
TypeInterface 181
FunctionTypeTerminal 580
HWTypeTerminal 91
StructureElement 50
DeviceMapping 10
DeviceType 4
BusCAN 3
BusSegment 3
MappedFunction 108

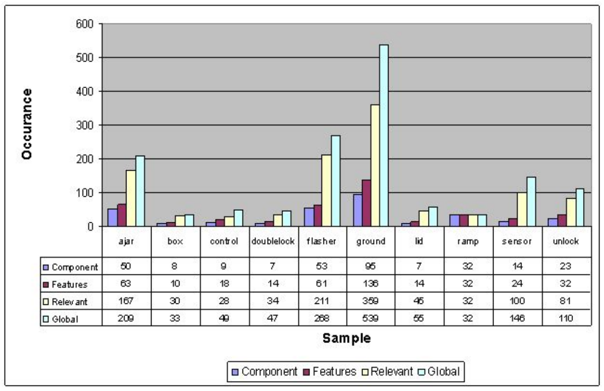
Attributes 12 / 13353

Name(Attribute) 4011
Name 3489
LongName 0
DEScription 0
ConnectionSegment 537
SourceTerminal 538
SinkTerminal 538
Interface 292
CompoundTerminal 269
HWTerminal 292
SWTerminal 302
Input 1543
DataType 1542

Figure 9. Dataset summary of project using ESCAPE design tool.

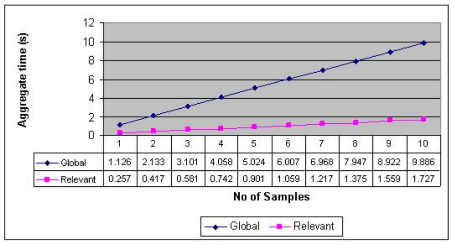
Three different approaches were adopted to evaluate and determine the performance with respect to matches and time.

- **Evaluation using a single element specification set** The first experiment was conducted on a single element specification set. A group of ten sets formed the input to determine the result set in both comprehensive (global) search and selective search as illustrated in Figure 10. The notion of comprehensive search is used, when scanning all occurrences of the specification set within projects, irrespective of whether they are components or features of those components. This can return a result set that contains false matches.



	ajar	box	control	doublelock	flasher	ground	lid	ramp	sensor	unlock
Component	50	8	9	7	53	95	7	32	14	23
Features	63	10	18	14	61	136	14	32	24	32
Relevant	167	30	28	34	211	359	45	32	100	81
Global	209	33	49	47	268	539	55	32	146	110

Figure 10. Occurrence graph for a single element specification set.



	1	2	3	4	5	6	7	8	9	10
Global	1.126	2.133	3.101	4.058	5.024	6.007	6.968	7.947	8.922	9.886
Relevant	0.257	0.417	0.581	0.742	0.901	1.059	1.217	1.375	1.559	1.727

Figure 11. Time graph for a single element specification set.

The pattern of the results displayed similar behavior.

Observations

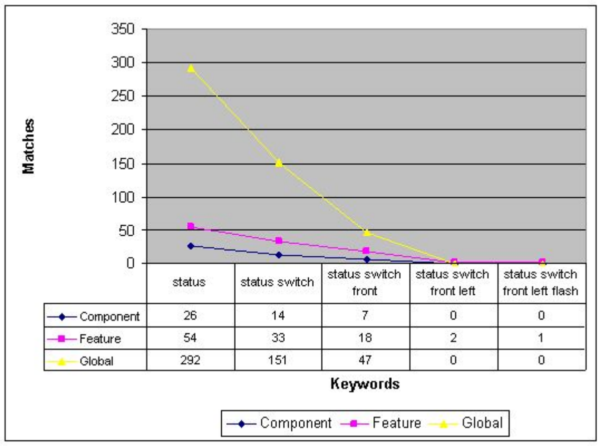
  - The comprehensive search yields a result set that contains every occurrence of the specification set, even if these nodes do not characterize a component.
  - The nodes representing components yield a result set which is somewhat realistic, though these do not epitomize the complete set desired. This is often observed when the component nodes do not match, but their features collectively match the specification set.
  - These nodes along with the feature set yield a more elaborate result set. A match contained by any node in a set of features would result in representing the component to which it belongs.

Figure 11 depicts the time taken to obtain the specification set illustrated in Figure 10. The time graph depicts the aggregate time required for global and selective search for a set of ten specification sets.

Observations

  - It is evident from these figures that the time required for comprehensive search exceeds the selective search - which is the method proposed in this article - by almost a factor of 5; this may be a dominant factor for large specification sets.

- **Evaluation using multiple element specification set** The second experiment was conducted using one up to seven element specification sets as a group illustrated in Figure 12.



	status	status switch	status switch front	status switch front left	status switch front left flash
Component	26	14	7	0	0
Feature	54	33	18	2	1
Global	292	151	47	0	0

Figure 12. Occurrence graph for multiple element specification sets.

Observations

  - The comprehensive search often yielded large result sets, as it searches in individual nodes that are treated as atomic.
  - The exhibited behavior is similar to the varying size of the specification set. As observed in Figure 12, the selective component-feature search result set demonstrates a value when the size of specification set exceeds 3, because in this case the matches take place across the boundary of the feature within the component. On the other hand the other methods return null result set as the search is only within the boundary of the element.
  - For any given size of specification set, the selective component-feature search returns a much smaller result set and is more precise.
  - Convergence is optimal with a specification set of size 3. If the size of the specification is too large the result may be null for both methods as shown in Figure 12.

- **Evaluation using different starting points for elements in specification sets**

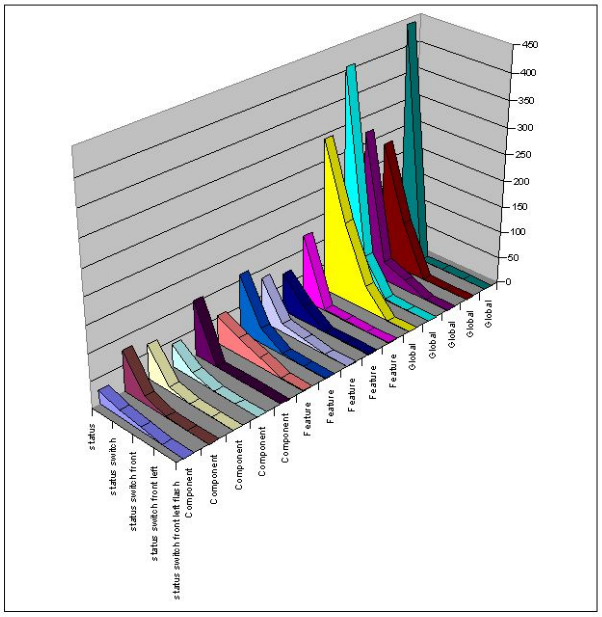


Figure 13. Occurrence graph for different starting points.

The third experiment was conducted searching for elements within specification sets using different starting points. Figure 13 depicts the result sets in comprehensive search and selective search.
To determine the effect of different starting points, a multiple-element specification set was used, where the orders of the elements were changed to obtain five sets. The result set for this exhibits the same pattern as the two experiments above.

VII. Conclusion

Managing variants is of utmost importance in today's large software bases as they reflect legal constraints, marketing decisions, and development cycles. As these software bases often grew from different sources and were developed by different teams using different tools it is in many cases very complicated if not nearly impossible to find artefacts that might be variants, both for historical reasons as for development purposes.

Searching algorithms have to reflect both the capability to match keywords and to reflect the structure that characterizes a component. Our proposed method is capable of both aspects and therefore helps the developer to find matches even in large and heterogeneous databases. In addition to that not only the required time for the search is a lot shorter, but also accuracy of the retrieved set of candidates is highly improved.

The developed prototype is itself independent of a specific tool as it works on textual descriptions that typically are available in XML.

References

[1] Ebert, C. and Salecker, J.; *Guest editors' introduction: Embedded software technologies and trends*. Software, IEEE, Vol 26(3): pp. 14-18, 2009

[2] Tanenbaum, A.S. and van Steen, M.; *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice Hall, 2006

[3] Clements, P. and Northrop, L.; *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2007

[4] Ebert, C. and Jones, C.; *Embedded software: Facts, figures, and future*. Computer, IEEE Vol 42(4): pp. 42-52, 2009

[5] Gomaa, H. and Webber, D.L.; *Modeling Adaptive and Evolvable Software Product Lines Using the Variation Point Model*. The Proceedings of the 37th Hawaii international Conference on System Sciences, 2004

[6] Oliveira, E., Gimenes, I., and Maldonado, J.; *A variability management process for software product lines*. CASCON 2005, The conference of the Centre for Advanced Studies on Collaborative research: pp. 225 - 241

[7] Heymans, P. and Trigaux, J.; *Software product line: state of the art*. Technical report for PLENTY project, Institut d'Informatique FUNDP, Namur, 2003

[8] Galster, M. and Avgeriou, P.; *Handling variability in software architecture: Problem and implications*. WICSA 2011, Ninth Working IEEE/IFIP Confernce on Software Architecture: pp. 171-180

[9] PRODUCT LINE HALL OF FAME; *"http://splc.net/fame.html"*. retrieved: 04,2012

[10] Bachmann, F. and Clements, P. C.; *Variability in Software Product Lines*, Technical Report -CMU/SEI-2005-TR-012, 2005.

[11] Bosch, J.; *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*, Addison-Wesley, 2000

[12] Burgareli, L.A., Selma, Melnikoff, S.S., and Mauricio Ferreira, G. V.; *A Variation Mechanism Based on Adaptive Object Model for Software Product Line of Brazilian Satellite Launcher*, ECBS-EERC 2009, First IEEE Eastern European Conference on the Engineering of Computer Based Systems: pp. 24-31

[13] IEEE; *Iso/iec standard for systems and software engineering - Rrecommended practice for architectural description of software-intensive systems*. Technical report, IEEE, 2000

[14] Atkinson, C. and Kühne, T.; *A generalized notion of platforms for model-driven development*. Model-Driven Software Development, Springer-Verlag, Berlin: pp. 119–136, 2005

[15] Frank, A.A. and Brenner, E.; *Model-based Variability Management for Complex Embedded Networks*. ICCGI 2010, The Fifth International Multi-conference on Computing in the Global Information Technology: pp. 305-309

[16] Frank, A.A. and Brenner, E.; *Strategy for modeling variability in configurable software*. PDES 2010, The 10th IFAC workshop on Programmable Devices and Embedded Systems

[17] ESCAPE; *"http://www.gigatronik2.de/index.php?seite=escape_produktinfos_de&navigation=3019&root=192&kanal=html"*. retrieved: 04,2012

[18] AUTOSAR; *"http://www.autosar.org/download/conferencedocs/03_AUTOSAR_Tutorial.pdf"*. retrieved: 04,2012