Jerzy Leyk

Normalised Frame Language – towards a Complete Metatext Execution Tree

1. Introduction

The frame is regarded as a cognitive model, a pattern that is applied, and simultaneously adapted, to a new, yet similar situation. In computing, such a concept of frames was introduced by Paul G. Bassett in the mid 1980 s[1]. There are two main reasons for applying this concept in software engineering:

a) to generate target language text (e.g. program code) in different variants depending on a predefined number of parameters and conditions. In other words, to obtain diversity based on structural similarity;
a) to discover similarities in an already existing code to reduce redundancy and identify potential errors[2].

In both cases the business purpose is to reduce the costs of software development and maintenance. The frame approach results also in higher clarity of created texts (e.g. software packages) and lowers the error proneness. It differs from other techniques known in software engineering. For example, object oriented programming (OOP) focuses on the construction of encapsulated functional units in the source code, whereas frames operate on texts of any kind – not just software codes (including OOP texts) – and organise them[3]. In OOP, the decisions are bound at run time, in the frames approach – at development time.

1 P.G. Bassett, *Framing Software Reuse: Lessons from the Real World*, Yourdon Press, Prentice Hall, 1997. Bassett's frames discussed in the paper should not be confused with frames of knowledge representation (ontology) languages (cf. the survey in P. Karp, *The Design Space of Frame Knowledge Representation Systems*, "Technical Note 520", Artificial Intelligence Center, SRI International, 1992, available at: http://www.ai.sri.com/pub_list/236, as of 2014–06-30).

2 H.A. Basit, S. Jarzabek, *Data Mining Approach for Detecting Higher-level Clones in Software*, "IEEE Transactions on Software Engineering", vol. 36, no 4, July/August 2009, pp. 497–514.

3 A comparative study of other techniques is beyond the scope of this paper. Pre-processors, object oriented programming, features oriented programming, compositional programming, software product lines may be mentioned, cf. *Framework for Software Product Line Practice*, Version 5.0, Software Engineering Institute, Carnegie Mellon, available at: http://www.sei.cmu.edu/productlines/frame_report, as of 2014–06–30.

The application of frames requires means that define frames and construct outputs using them – frame languages. Many different frame languages have already been developed and implemented[4]. Despite their different forms, the main instruments have similar motivation and functionality.

Documented theoretical foundations of frame languages are poor[5]. The authors are much more oriented on the engineering aspects. This paper focuses on some of the foundations. The study of frame languages in the paper is motivated by the following main questions:

1. What are the properties of all possible final texts that can be generated using a given set of frames?
2. What can be proposed in order to establish measures for quantifying and qualifying such a set?

As all of the possible final texts available from a set of frames constitute, for example, a software system, from the engineering point of view it becomes crucial to estimate it and identify its weaknesses.

The first step to achieve these objectives requires a theoretical background that will be independent from different realisations of frame languages. The second step – a proposal of measures as such. The paper concentrates on the first step.

2. Definitions

The meaning of terms presented here is not a formal one. The definitions below should help to understand the terms as they are used in the paper:

---

[4] Cf. P.G. Bassett, *op.cit.*; with regard to XVCL: S. Jarzabek and H. Zhang, *XML-based Method and Tool for Handling Variant Requirements in Domain Models*, "Proceedings of the 5th International Symposium on Requirements Engineering", RE'01, Toronto 2001, pp. 166–173, as well as: *XVCL: A Tutorial*, http://xvcl.comp.nus.edu.sg/xvcl_tutorial.php, Singapore, 2010; with regard to Scriptor: J. Leyk, *Frame Technology Applied in the Domain of IT Processes Job Control*, "Advanced Information Technologies for Management – AITM 2011: Intelligent Technologies and Applications", eds. J. Korczak, H. Dudycz, M. Dyczkowski, Research Papers of Wroclaw University of Economics no. 206, Wrocław 2011, pp. 96–107; with regard to FPL: F. Sauer, *Metadata driven multi-artifact code generation using Frame Oriented Programming*, Position Paper, OOPSLA 2002 Workshop on Generative Techniques in the context of Model Driven Architecture, November 4~8, 2002, available at: http://www.volantec.biz/metadata-driven.pdf, as of 2014–06–30.

[5] For instance P.G. Bassett, *op.cit.*, and Ch. Holmes and A. Evans, *A review of Frame Technology*, University of York, 2003, available at: http://www.cs.york.ac.uk/ftpdir/reports/YCS-2003–369.pdf, as of 2010–06–30.

Language[6] – a set of elements and rules which is used to compose sequences from these elements recognised as valid (correct).
Lexeme – a minimal unit of language that is not further decomposed into smaller units for the purpose of particular analysis.
Text – any sequence of lexemes in the given language.
Object language – a language in which texts from a particular field are supposed to be expressed.
Metalanguage[7] – a language the lexemes of which can be used in texts of an object language according to its own rules and which can be finally interpreted as sequences of object language lexemes. The metalanguage is object language independent.
FL – frame language – the metalanguage defined to compose texts in object language using the concept of frames.
NFL – Normalised Frame Language – the frame language defined in this paper as the common basis for a theoretical analysis of metatexts.
AFL – application specific frame language – any FL designed to be used in a particular domain, among them the **classical FL** as presented by Bassett[8].
Metatext, generic text – a text in the frame language including lexeme sequences from an object language. Such a mixture implies that metalanguage lexemes need to be recognisable as not belonging to the object language[9].
Text processor – a device (program, process, actor or similar) capable of creating an object language text from any metatext built by the use of a metalanguage.
Metatext execution, text generation – the process in which frame language elements in the metatext are executed (interpreted) conforming to the frame language principles in order to obtain an object language text.
Final text – text in the object language resulting from the metatext execution.
Frame – a uniquely identified structural fragment of metatext from which the text processor may generate different variants of object language texts by means of frame adaptation rules. Conceptually, a frame is designed to be reusable.

---

[6] No further assumptions on elements nor rules are relevant for the discussion on frames, therefore no formal definition of the term *language* is applied.

[7] Definitions of a metalanguage usually describe a metalanguage as a language in which properties, rules, statements, etc. about an object language are expressed. The definition presented here has another purpose.

[8] A study on such implementations may be found in Ch. Holmes and A. Evans, *op.cit*.

[9] The metatext should satisfy the frame language syntax. The object language syntax is irrelevant here; it becomes crucial in the final texts created. Hence, the author of the generic text must make sure that final outputs satisfy the syntax rules of the object language.

Directive, command – a formal structure of the frame language that describes a valid operation for the text processor and is represented by a sequence of metatext lexemes terminated by the begin and end markers.
Frame parameter – a metalanguage lexeme that is replaced by the processor at the moment of the metatext execution by the currently assigned sequence of lexemes.
Reserved word –a predefined metalanguage lexeme applied in the metalanguage directive as part of its syntax. A reserved word in a metatext outside a directive is considered an object language lexeme.

3. Normalised Frame Language

FL elements in the metatext tell the text processor what actions (operations) are to be performed in order to obtain the final text in the object language. When calling (invoking) the processor, one must define which final text from the many possible ones (at least one) incorporated in the metatext is to be generated. The selection is done by setting the invocation parameters. If no parameter is set externally, the processor can create only one final text, otherwise a variant of the final text that conforms with the externally set values of frame parameters will be generated.

For the purpose of theoretical considerations an NFL is proposed below. The use of such common language provides some metatext theory foundations leading to the concept of CoMET. This objective requires a stronger discipline, reflected already on the level of the language, than in the classical FLs, where many NFL properties are, in fact, hidden in the FL processors. Therefore, it is assumed that a well-defined metatext based on the classical FL can be transformed into the NFL format[10]. Such an assumption justifies the use of the term 'normalised frame language'. Hence, the transformation can be viewed as normalisation.

[10] On the level of language syntax, such transformation possibilities of classical FLs (also breakpoints and loops) and Scriptor (also conditions and steps) seem evident. A proof of an equivalent transformation of generated results is, however, not fully possible as long as the properties of the text processor behind a particular AFL remain unknown or are not taken into account.

BNF-like notation rules are applied to the NFL syntax presentation. The reserved words and syntax details applied here are only illustrative. In practice, they may be represented in a metalanguage differently[11].

Table 1. Normalised frame language syntax

No.	Syntax	Function	Description
1a	FRAME <frame-name> FRAME-BEGIN <metatext-fragment> FRAME-END	Frame definition directive	Identifies a frame and specifies the frame body as <metatext-fragment>. <frame-name> must be unique within the metatext.
1b	MODULE <module-name> MODULE-BEGIN <module-body> MODULE-END where <module-body> is: [{<external-frame-parameter>} ...] [ADAPT <specification-class>] <metatext-fragment>	Module definition directive[12]	Identifies a frame that is allowed as the starting point for processor invocation (that is for final text population), specifies applicable external frame parameters declared by VARIABLE or SELECT and indicates the class of applicable specification frames. Within any metatext a module can be used exactly like a frame.
1c	SPECIFICATION <specification-name> CLASS <specification-class> [<specification-body>] SPECIFICATION-END where <specification-body> is: {<SET-directive>} ...	Specification frame definition directive	Identifies external frame parameter assignments valid for a particular execution of a module, provided this specification class is attached to the module. Within any metatext a specification can be used exactly like a frame.

[11] For example, so as to uniquely distinguish each directive, the initial reserved word can be preceded by a special character or string of characters; similarly, any reference to a frame parameter in the metatext may be preceded and followed by special symbols.

[12] In frame languages usually there is no distinction made between a module and a specification frame. The entire relevant frame body is placed in the specification frame which is invoked directly. Such approach means that either each specification frame redundantly repeats the text body placed in a module frame or ADAPTs a main control frame that works as a module. Cf. P.G. Bassett, *op.cit.*, J. Leyk, op.cit., F. Sauer, *op.cit.*, XVCL: A Tutorial, *op.cit.*

No.	Syntax	Function	Description
2a	VARIABLE <variable-name> {EXTERNAL | LOCAL | INHERITED | RETURN | RETURN- LOCAL} PHRASE-BEGIN <value-range> PHRASE-END	Declaration directive of variable type frame parameter	Declares a variable and the range of allowed values. Only a frame parameter of variable type can be used in parameter substitution.
2b	SELECTOR <selector-name> {EXTERNAL | LOCAL | INHERITED} PHRASE-BEGIN <value-range> PHRASE-END	Declaration directive of selector type frame parameter	Declares a selector and the range of allowed values. Only a frame parameter of the selector type may be used as a frame parameter in the CASE directive.
3	SET {<variable-name> | <selector-name>} PHRASE-BEGIN <parameter-value> PHRASE-END	Frame parameter assignment directive	Assigns value to a frame language parameter. The value will be substituted for the parameter once encountered by the text processor from this point according to the scoping attribute. <parameter-value> must be a lexeme from the <value-range> in the parameter declaration.
4	<variable-name> occurrence	Variable substitution	Replaces <variable-name> in <metatext-fragment> in its position by the currently assigned value. Not applicable within directives (except for <metatext-fragment>).
5	ADAPT <frame-name>	Frame adaptation and insertion directive	Adapts the indicated frame by executing NFL instruments within the frame and inserting the result in the ADAPT position. Module and specification frames may be applied, the latter only in a module. Nesting of ADAPTs is allowed, however non-recursively (acyclic).
	CASE <selector-name> IN {<range> PHRASE-BEGIN <metatext-fragment> PHRASE-END} … CASE-END	Disbranching directive	Inserts one of the <metatext-fragment>s depending on the <range> that is satisfied by the current value of the selector-name>. Assumptions on <range>s: a) ranges are subsets of the <value-range> in the selector's declaration, b) ranges are distinct,

No.	Syntax	Function	Description
			c) the number of ranges is finite, d) the sum of ranges equals the <value-range> (exhaustiveness). CASE directives can be nested but their <selector-name>s should not.
7	INVOKE <module-name> [SPECIFICATION <specification-name>] <invocation-body> INVOKE-END where <invocation-body> is: [{<SET-directive>} ...]	Processor invocation quasi-directive	INVOKE is not a part of the metatext. It tells the text processor which module should be traversed for final text generation and what the initial values of the frame parameters relevant for a particular run of the processor are.

Source: Own work.

Table 2. Normalised frame language – additional rules and constraints

No.	Subject	Description
1	Reserved words	Are expressed in the frame syntax in uppercase.
2	<names>	A name must not be a reserved word. Sets of <frame-name>s, <variable-name>s, <selector-names>s and <specification-class>s should be distinct (mutually exclusive) within one metatext.
3	<metatext-fragment>	A contiguous sequence within the metatext (an empty one as well). Both object language and metalanguage lexemes are allowed.
4	Frame declarations	May not be nested, i.e. <metatext-fragment> cannot include another FRAME, MODULE or SPECIFICATION directive.
5	Frame parameter types	Two types of frame parameters exist in NFL: a) variable – may be the subject of parameter substitution, b) selector – may be used in a CASE directive as <frame-parameter>. Values of frame parameters are regarded as one lexeme from an ordered set specified by <value-range> (0-length lexeme is always considered as valid). The way <value-range> is specified is irrelevant for this paper.

No.	Subject	Description
6	Frame parameter scoping	Frame parameter scoping attributes determine in which frame value can be assigned to the parameter and over which frames this assignment remains valid during metatext execution. a) INHERITED – valid assignment only in the frame where declared, the value inherited down in the hierarchy. This is the default option for any frame parameter declaration. b) EXTERNAL – may be declared only in a module, valid assignment only in this frame and in an INVOKE directive, the value inherited down in the hierarchy. The purpose of EXTERNAL is to predefine clearly which parameter can be set in INVOKE. c) LOCAL – much like INHERITED. LOCAL overwrites the validity and scope of a parameter of the same name declared higher in the hierarchy. LOCAL helps developing a metatext in independent pieces protecting coincidental use of the same <parameter-name> for different purposes. d) RETURN – valid assignments in any frame in the frame hierarchy, recently assigned value holds down and up in the hierarchy. RETURN serves the purpose of transferring values from frames in the upward direction and of modifying them down the hierarchy. e) RETURN-LOCAL – properties of RETURN and LOCAL.
7	INVOKE	INVOKE directive may be considered attached to the metatext at its very beginning as a separate module, where SET phrases become SET directives and the frame ends with an ADAPT of the invoked <frame-name>. The link between a specification frame, a module and the INVOKE directive is expressed in the following way: a) in the SPECIFICATION directive, the clause CLASS indicates a class of specification frames to which the specification frame belongs, b) in the MODULE directive, the ADAPT <specification-class> directive (when present) points to the class of specification frames that may be applied in the module, c) once indicated in the INVOKE, directive the <specification-name> frame is attached to the invoked module via the ADAPT <specification-class> directive, d) the frame parameters assigned in the specification frame must be declared in the module as EXTERNAL – otherwise no assignment will take place. In the INVOKE directive the list of SETs following the SPECIFICATION clause is executed after the specification frame contents – this allows overwriting *ad hoc* some specification frame assignments. The <specification-class> concept enables a better control over specification frames allowed for invocation via a particular module. Once specification frames can be individually added during metatext development, the module contents remains stable.

Source: Own work.

4. Metatext theory

Evaluation of the properties of FL application requires the establishment of certain measures. Many authors (for instance Bassett and Sauer) who deal with FL present frame structures using graphs that reflect the relations between frames via ADAPT directives only and that allow multiple parents. This is definitely insufficient when it comes to treating and measuring the potential of a metatext, first of all because CASE directives are hardly ever represented in such an approach.

CoMET – a complete metatext execution tree based on NFL is introduced below. The objective is to represent a particular module of metatext with all possible variants for execution. The collection of all modules of the given metatext, not excluding standalone frames, is then a graph theory forest.

4.1. CoMET – complete metatext execution tree

Let G be a graphical presentation of a well formed[13] NFL module achieved according to the rules in table 3.

Table 3. CoMET construction rules

No	Rule description
1	Metatext sequences of lexemes are grouped and its nodes denoted as follows: a) consecutive sequence of object language lexemes and/or frame variables within the <metatext-fragment>, not including or being a part of ADAPT or CASE directive – denoted as <frame-name>. OL-<sequence number in the frame>[14] b) ADAPT directive with its body – denoted as <frame-name> c) CASE directive with its body – denoted as <frame-name>. CASE-<CASE directive number in the frame> d) <metatext-fragment> of a single range phrase of a CASE directive – denoted as <frame-name>. CASE-<CASE directive number in the frame>. R-<range number in this CASE>. Other lexemes are not considered – e.g. SET directives, frame parameter declaration directives, frame definition lexemes beyond the <metatext-fragment> being their body.

[13] 'Well formed' means here correct from the point of view of the grammar of the metalanguage only, not of the object language.

[14] On the metalanguage level, one can imagine a single OL-sequence as a uniquely identified object language piece of a document including (or not) FL variables to be replaced by object language lexemes. From such point of view, the metatext strongly resembles an editing tool to manage the composition of documents from smaller reusable pieces.

No	Rule description
2	Root node is the body of the module definition, i.e. its <metatext-fragment>.
3	Edges are ordered pairs of nodes of the following types: a) (metatext-fragment, OL-sequence within the metatext-fragment) b) (metatext-fragment, ADAPT directive within the metatext-fragment) c) (metatext-fragment, CASE directive within the metatext-fragment) d) (CASE directive, metatext-fragment of its single range phrase) e) where metatext-fragment is the body of a frame or of a single range phrase of the CASE directive.
4	The following NFL syntax rules apply: a) nesting of ADAPT directives may not lead to frame recursion (cycle), b) list of ranges per each CASE directive is finite and exhaustive, c) nesting of CASE directives may not lead to <selector-name> repetition.
5	To create a graph, the module definition body is processed sequentially, each encountered ADAPT is resolved like the module frame, CASE directive split by ranges, and so on, until all the edge ends become OL-sequence leaves.

Source: Own work.

Figure 1 presents a sample of a single module metatext and figure 2 its graphical representation.

Some features and properties of the complete metatext execution of graph G:

1. The construction process of graph G is finite due to the following principles: (a) the metatext is finite, (b) the number of lexeme sequences, ADAPTs, CASEs and CASE-range occurrences are finite, (c) ADAPTs are not cycled.
2. Nodes are not unique, but there are no cycles. Different occurrences of the same frame label with the entire sub-tree are repeated once reused.
3. In the construction of final texts, the relation between edges from CASE to CASE ranges is treated as an alternative (OR); otherwise as a conjunction (AND).
4. Leaves are <metatextfragment>s – empty or composed of OL-sequences only.
5. Any final text available via the metatext is a subset of the sequence of leaves.
6. A sequence of leaves may be presented in a bracketed form preserving the inherited AND/OR relations between the edges on the path from the root.
7. Not all frames of the metatext are present in the graphical representation – those for which there is no path from the root are absent.

A linear representation on leaves level using $\wedge$ (AND – concatenation), $\vee$ (OR – alternative), (,) of the tree:

A. OL-1 $\wedge$ B. OL-1 $\wedge$ (A. OL-2 C. OL-1 $\wedge$ B. OL.1) $\wedge$ (B. OL.1 $\vee$ empty) $\wedge$ B. OL-1

where: empty stands for 0-length lexeme.

```
MODULE A MODULE-BEGIN
    SELECTOR parameter-1 EXTERNAL PHRASE-BEGIN val-1, val-2 PHRASE-END
    SELECTOR parameter-2 EXTERNAL PHRASE-BEGIN val-3, val-4 PHRASE-END
    ADAPT specification-class-1
    VARIABLE variable-1 LOCAL PHRASE-BEGIN a, b, c PHRASE-END
    OL-sequence-A1
    SET variable-1 PHRASE-BEGIN a PHRASE-END
    ADAPT B
    CASE parameter-1 IN
        {val-1}  PHRASE-BEGIN
                         OL-sequence-A2
                         INSERT C
                 PHRASE-END
        {val-2}  PHRASE-BEGIN
                         CASE parameter-2 IN
                         {val-3}  PHRASE-BEGIN ADAPT B PHRASE-END
                         {val-4}  PHRASE-BEGIN PHRASE-END
                         CASE-END
                 PHRASE-END
    CASE-END
    SET variable-1 PHRASE-BEGIN b PHRASE-END
    ADAPT B
MODULE-END

SPECIFICATION spec-1 CLASS specification-class-1
    SET parameter-1 PHRASE-BEGIN val-1 PHRASE-END
PHRASE-END

FRAME B PHRASE-BEGIN
    OL-sequence-B1-with-substitutions-of-variable-1
PHRASE-END

FRAME C PHRASE-BEGIN
    OL-sequence-C1
    SET variable-1 PHRASE-BEGIN c PHRASE-END
    INSERT B
PHRASE-END

FRAME D PHRASE-BEGIN
    OL-sequence-D1
    INSERT B
PHRASE-END
```

Figure 1. Example of a metatext

Source: Own work.

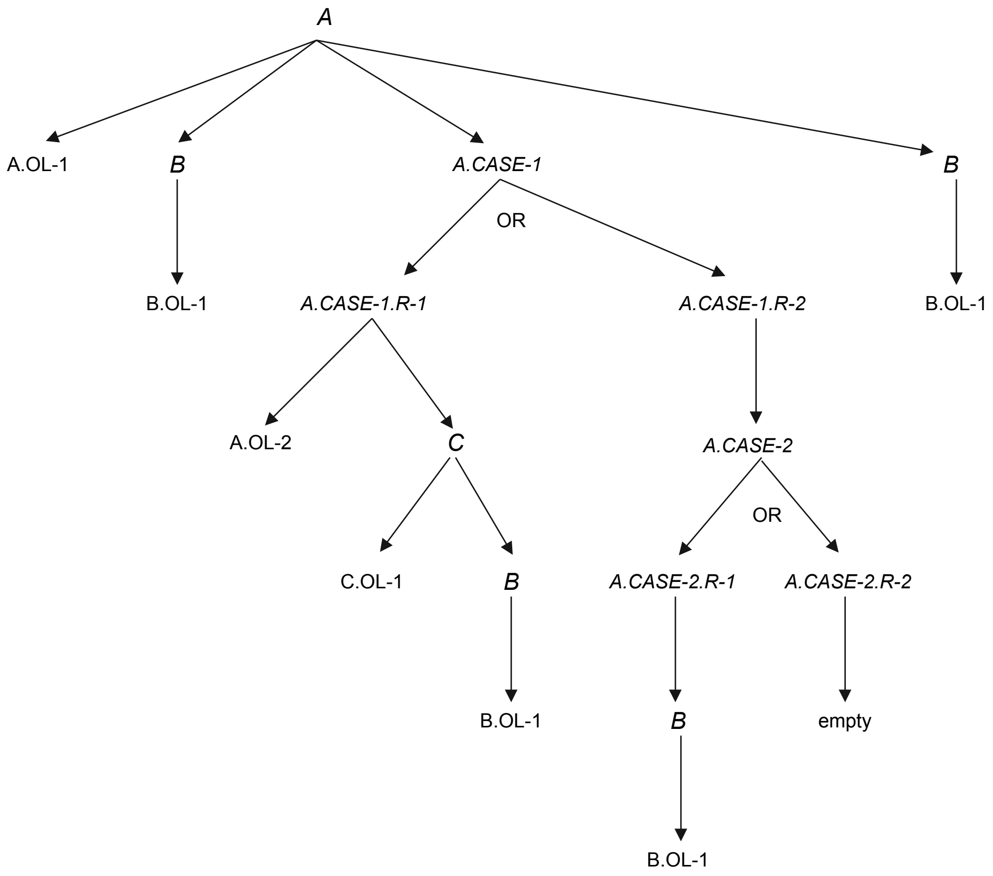

Figure 2. Tree representation of a metatext – CoMET example
Source: Own work.

4.2. Metatext and the graph theory

A tree as a formal structure is crucial for analysing issues related to metatext measurement. In the graph theory, the following definition of tree is applied[15]:

A **tree** (undirected) is an undirected graph G that is simple, is connected, and has no cycles, and where:

1. An undirected **graph** is an ordered pair $G = (V, E)$ comprising a set V of **vertices** (nodes) together with a set E of **edges** (lines), which are two-element

[15] R. Diestel, *Graph Theory*, Springer–Verlag, Berlin 2005.

unordered subsets of *V* (i.e. each edge is an unordered pair of two vertices and this relation is considered a connection between the two vertices).

2. A **simple graph** is an undirected graph that has no loops (i.e. each edge is a pair of distinct nodes) and no more than one edge between any two different vertices (i.e. edges as pairs in E are distinct).
3. A graph is **connected**, if every pair of vertices in the graph is connected, i.e. for any pair of vertices *u*,v there is a path from *u* to *v*, where a **path** is a sequence of vertices such that from each of its vertices there is an edge to the next vertex in the sequence.
4. A **cycle** is a path where the start vertex and the end vertex are the same.

A graph or a simple graph is **directed**, if the edges of E are ordered sets, i.e. the relation represented by an edge points from one vertex (begin, source, predecessor) to the other one (end, target, successor). A **directed acyclic graph** (**DAG**) is a directed graph with no directed cycles.

It may be easily proven that G is a DAG by virtue of its construction principles mentioned in 4.1: $G = (V, E)$, where, *V* are nodes (p. 1), *E* (p. 3) are two-element ordered subsets of *V* and there are no cycles (p. 4). It may also be proven that from the algebraic point of view G is a semilattice[16] bounded by the root.

5. Conclusions

Frame languages and frame technology are not widely known in the software community. Lack of sound theoretical basis may be one of the obstacles in getting wider interest. This paper presents some fundamental considerations. These considerations include the author's new concept of a normalised frame language, metatext normalisation and a complete metatext execution tree (CoMET) based on the graph theory. Further studies into the nature of frame languages are planned. The challenge for FL implementations is not just a pure set of language instruments, but an advanced software environment which will be able to deal with FL metrics, metatext normalisations, discovering weak solutions, supporting testing and debugging. Creation of metrics deriving from the concept

[16] An algebraic structure is a lattice if and only if there are two binary operations in the set V that are linked by the absorption low [9]. In the complete metatext execution tree, there are no such operations. Once a root is with $\wedge$ operation considered as the meet, there is no join and vice-versa.

of CoMET, may be the next step towards metatext quality measurement and verification. I hope the paper will also encourage to develop frame technology software for non-software application domains.

6. Acknowledgments

I want to thank Stan Jarzabek for his valuable feedback on the idea of normalised frame language presented in this paper and the impact of FL foundation oriented research on practical FL implementations.

References

Basit, H. A., Jarzabek, S., *Data Mining Approach for Detecting Higher-level Clones in Software*, "IEEE Transactions on Software Engineering", vol. 36, no. 4, July/August 2009, pp. 497–514.

Bassett, P. G., *Framing Software Reuse: Lessons from the Real World*, Yourdon Press, Prentice Hall, 1997.

Diestel, R., *Graph Theory*, Springer-Verlag, Berlin 2005.

Framework for Software Product Line Practice, Version 5.0, Software Engineering Institute, Carnegie Mellon, available at: http://www.sei.cmu.edu/productlines/frame_report, as of 2014–06–30.

Jarzabek, S. and Zhang, H., *XML-based Method and Tool for Handling Variant Requirements in Domain Models*, "Proceedings of the 5th International Symposium on Requirements Engineering", RE'01, Toronto 2001, pp. 166–173.

Holmes, Ch., Evans, A., *A review of Frame Technology*, University of York, 2003, available at: http://www.cs.york.ac.uk/ftpdir/reports/YCS-2003–369.pdf, as of 2010–06–30.

Karp, P., *The Design Space of Frame Knowledge Representation Systems*, "Technical Note 520", Artificial Intelligence Center, SRI International, 1992, available at: http://www.ai.sri.com/pub_list/236, as of 2014–06–30.

Leyk, J., *Frame Technology Applied in the Domain of IT Processes Job Control*, "Advanced Information Technologies for Management – AITM 2011: Intelligent Technologies and Applications", eds. J. Korczak, H. Dudycz, M. Dyczkowski, Research Papers of Wroclaw University of Economics no. 206, Wrocław 2011, pp. 96–107.

Maclane, S., Birkhoff, G., *Algebra*, The Macmillan Company, New York 1967.

Sauer, F., *Metadata driven multi-artifact code generation using Frame Oriented Programming,* Position Paper, OOPSLA 2002 Workshop on Generative Techniques in the context of Model Driven Architecture, November 4–8, 2002, available at: http://www.volantec.biz/metadata-driven.pdf, as of 2014–06–30.

XVCL: A Tutorial, http://xvcl.comp.nus.edu.sg/xvcl_tutorial.php, Singapore, 2010.

* * *

Znormalizowany język ramek i pełne drzewo realizacji metatekstu

Streszczenie: Koncepcja ramek jako instrumentów poznawczych modelujących rzeczywiste sytuacje znana jest także w inżynierii oprogramowania. Języki do tworzenia ramek i ich zastosowania generujące podobne, lecz różne wyniki końcowe datuje się na połowę lat 80. XX w. W artykule przedstawiono kilka teoretycznych podstaw tego rodzaju języków opartych na implementacjach zrealizowanych w praktyce. Zdefiniowany został znormalizowany język ramek wraz ze szczegółowym przedstawieniem jego środków językowych. Na tej bazie, przy wykorzystaniu teorii grafów, została rozwinięta koncepcja CoMET – pełnego drzewa realizacji metatekstu. Znormalizowany język ramek oraz koncepcja CoMET umożliwiają badanie i tworzenie rozwiązań, które mogą mierzyć i porównywać właściwości metatekstów z różnych punktów widzenia. Podstawy teoretyczne przedstawione w artykule mają na celu wsparcie inżynierów oprogramowania w pełniejszym zrozumieniu natury języków ramek oraz zachęcenie do tworzenia pełnych środowisk rozwojowych przeznaczonych także do dziedzin innych niż rozwój oprogramowania.

Słowa kluczowe: języki ramek, technologia ramek, rozwój oprogramowania