Dyna
ISSN: 0012-7353
dyna@unalmed.edu.co
Universidad Nacional de Colombia
Colombia

Bolaños-Castro, Sandro J.; González-Crespo, Rubén; Medina-García, Victor H.; Barón-Velandia, Julio
Conceptual framework language – CFL –
Dyna, vol. 81, núm. 185, junio, 2014, pp. 124-131
Universidad Nacional de Colombia
Medellín, Colombia

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MEDELLÍN
FACULTAD DE MINAS

DYNA

http://dyna.medellin.unal.edu.co/

Conceptual framework language – CFL – Lenguaje de marcos conceptuales – LMC –

Sandro J. Bolaños-Castro [a], Rubén González-Crespo [b], Victor H. Medina-García [c] & Julio Barón-Velandia [d]

[a] *Facultad de Informática, Universidad Distrital Francisco José de Caldas, Colombia. sbolanos@udistrital.edu.co*
[b] *Escuela de Ingeniería, Universidad Internacional de La Rioja, España. ruben.gonzalez@unir.net*
[c] *Facultad de Informática, Universidad Distrital Francisco José de Caldas, Colombia. vmedina@udistrital.edu.co*
[d] *Facultad de Informática, Universidad Distrital Francisco José de Caldas, Colombia. jbaron@udistrital.edu.co*

Received: February 8th, 2013. Received in revised form: February 27th, 2014. Accepted: April 22th, 2014

Abstract
This paper presents the Conceptual Frameworks Language –CFL–, it aims to bridge the gap between programming languages and design languages, using the mechanism of schematizing, this approach changes the complexity of the syntax of programming languages and complexity of the diagramming for ease of assembly and nesting of frames or conceptual blocks like Lego, we present the possibilities offered by CFL as a Language nearer to solving problems using computational and scientific vocabulary, which is transparent to the user, we outline comparisons and integrations with languages like java and UML, we propose metrics and develop the platform in java language.

Keywords: Language, scheme, metrics, contextualization, abstraction, vocabulary, syntax, semantics

Resumen
Este artículo presenta el Lenguaje de Marcos Conceptuales –LMC–, su objetivo es cerrar la brecha entre los lenguajes de programación y los lenguajes de diseño, empleando el mecanismo de la esquematización, este propone cambiar la complejidad de la sintaxis de los lenguajes de programación y la complejidad de la diagramación por la facilidad de ensamble y anidamiento de marcos o bloques conceptuales a manera de Lego, se presenta las posibilidades que brinda LMC como un leguaje más próximo a la resolución de problemas empleando un vocabulario computacional y científico, que se hace transparente al usuario, se plantean comparaciones e integraciones con lenguajes como java y UML, se proponen métricas y se hace una implementación de la plataforma en lenguaje java.

Palabras clave: Lenguaje, esquema, métricas, contextualización, abstracción, vocabulario, sintaxis y semántica.

1. Introduction

It is necessary to propose computer languages approaching human languages, making some unintuitive computational concepts transparent. Despite the diversity of programming and modeling languages, they care little about issues like:

- Provide a simple and intuitive representation.
- Talk a less computational language.
- Facilitate Direct Model Execution tracing.

The language should be the vehicle of abstraction; it should be simple, robust and complete to be more robust. This is achieved by hiding its complexity levels, using layers. The first layer covers a formal level which supports and extends mechanisms already developed and recognized such as encapsulation, security, generality, reuse, among others [1]. The second layer covers a particular level ease of use, focused on approaching the model and reality, taking in account the human thinking model.

To address these concerns, the paper presents the CFL language, its grammar, the comparison between schema and diagram, principles and metrics, closing with the implementation of the platform, case studies, and conclusions.

2. Conceptual Framework Language –CFL–

CFL, is a modeling language focused on abstraction and contextualization of knowledge. See Fig. 1.

When communicating an idea, this language can be used in both ways, spoken or written. Treasures such as the Rosetta Stone [2], unveiled a past steeped in pictograms; rock art paintings tell about the lives of our ancestors, in images that constitute a simple but expressive language.

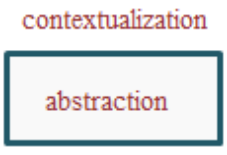

Figure 1: Contextualization and Abstraction

The language can range from an informal expression, captured in an image, to the rigorous expression represented in a word or syntactic structure. The CFL proposal is to exploit the power of formal language with the power of symbolic schematization, for that, it proposes two concepts, abstraction and contextualization. The abstraction mechanism extracts the essential characteristics, using cognitive models like: paradigms, values, principles and behaviors; abstraction uses introspection as strategy which defines the search mechanisms stock of knowledge in the same individual.. Moreover, contextualization uses an observation scheme to find the answers in external phenomena. The strategy used is immersion, which in contrast to introspection, seeks to reason about the phenomena through the use of external structures.

Contextualization and abstraction are the basis for CFL construction, in which individual and collective knowledge are contrasted. Paradigms as "structured", "object-oriented", including "declarative models" are based on abstraction [3], losing the potential of immersion, useful in solution modeling.

3. Grammar in –CFL–

Next, the concepts of grammar, derivation (production) and language are defined, all primordial for CFL formalization.

A phrase structure grammar $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{T}, \boldsymbol{S}, \boldsymbol{P})$ consists of a vocabulary $\boldsymbol{V}^*$, a subset $\boldsymbol{T}$ of $\boldsymbol{V}^*$ formed by the terminal elements, and a initial symbol $\boldsymbol{S}$ of $\boldsymbol{V}-\boldsymbol{T}$ and a set $\boldsymbol{P}$ of productions. The $\boldsymbol{V}-\boldsymbol{T}$ set is denoted by $\boldsymbol{N}$. The elements of $\boldsymbol{N}$ are called nonterminal elements [4].

Furthermore, a vocabulary $\boldsymbol{V}$ (or alphabet) is a finite and non empty set, whose elements are called symbols, a word in $\boldsymbol{V}$ is a finite string of $\boldsymbol{V}$ elements. The empty word or empty string denoted by $\boldsymbol{\lambda}$ is the string without symbols. The set of all words about $\boldsymbol{V}$ is denoted by $\boldsymbol{V}^*$. A language in $\boldsymbol{V}$ is a subset of $\boldsymbol{V}^*$ [4].

Another important definition is the derivation: $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{T}, \boldsymbol{S}, \boldsymbol{P})$ is a grammar with sentence structure. Also $\boldsymbol{w_0} = \boldsymbol{lz_0 r}$ (this is a *concatenation* $\boldsymbol{lz_0}$ **y** $\boldsymbol{r}$) and $\boldsymbol{w_1} = \boldsymbol{lz_1 r}$ about $\boldsymbol{V}$. If $\boldsymbol{z_0} \rightarrow \boldsymbol{z_1}$ is a production of $\boldsymbol{G}$, we say that $\boldsymbol{w_1}$, is directly derived from $\boldsymbol{w_0}$, and we write $\boldsymbol{w_0} \Rightarrow \boldsymbol{w_1}$. If $\boldsymbol{w_0}, \boldsymbol{w_1}, \ldots \boldsymbol{w_n}$ are strings about $\boldsymbol{V}$ such that $\boldsymbol{w_0} \Rightarrow \boldsymbol{w_1}$, $\boldsymbol{w_1} \Rightarrow \boldsymbol{w_2} \ldots, \boldsymbol{w_{n-1}} \Rightarrow \boldsymbol{w_n}$ we say that $\boldsymbol{w_n}$ is derivable or is derived from $\boldsymbol{w_0}$ and will be denoted $\boldsymbol{w_0} \overset{*}{\Rightarrow} \boldsymbol{w_n}$. The sequence of steps used to obtain $\boldsymbol{w_n}$ from $\boldsymbol{w_0}$ is called derivation [4].

Finally, the language generated by $\boldsymbol{G}$ (or the $\boldsymbol{G}$ language) must be defined, denoted by $\boldsymbol{L}(\boldsymbol{G})$, as the set of all terminal strings derived from initial state $\boldsymbol{S}$, Equation 1. [4].

$$\boldsymbol{L}(\boldsymbol{G}) = \left\{ \boldsymbol{w} \in \boldsymbol{T}^* \mid \boldsymbol{s} \overset{*}{\Rightarrow} \boldsymbol{w} \right\} \quad (1)$$

3.1. Productions of CFL in BNF

The Backus Naur form was initially created in order to define the syntactic structure of algol60 programming language [5]. BNF defines the syntactic structure of the language. CFL has the following syntactic structures:

<conceptual framework> ::= <frame><concept>
<frame> ::= <closed border> |
<open border> |
< semi closed border >
<concept> ::= <criteria> |'<archetype>'
<criteria> |'< archetype >'
<criteria><separator><body>
<criteria> ::= <definition>|<inquiry>|
<proof>|<elaboration>
<definition> ::= <variable>|<statement >|
<free text>
<inquiry> ::= <logic utilization><?>
<proof> ::= <verification action><!>
<elaboration> ::= <user extension>
<archetype> ::= λ|<property><name>|
<property><name><:>
<category>|<property><name>:
<category>(<interaction>)
<separator> ::= <sequential>|<parallel>
<body> ::= λ|<conceptual framework>|
<body>< conceptual framework>

4. CFL as a Languague

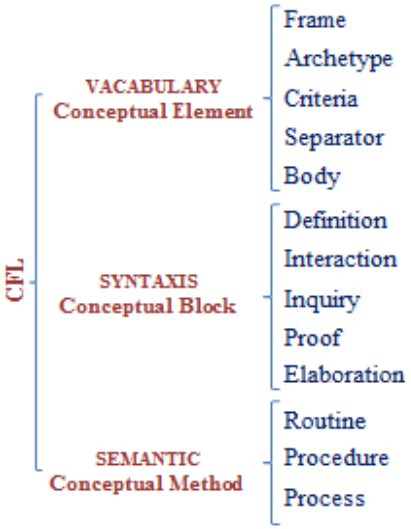

Figure 2: CFL as a Languague

CFL, Fig. 2, is formed by: a vocabulary or conceptual element, syntax or conceptual block and semantics or conceptual method.

4.1. Vocabulary of CFL

CFL is constituted by both frame and concept

Frame: is the frontier that separates a specific concept of the universe of discourse properly contextualized according to the domain proposed. See Fig. 3.

Figure 3: Frame

Concept: Set the domain of knowledge to be extracted from the universe of discourse. A concept consists of an archetype, criterion, a separator and a body. See Fig. 4.

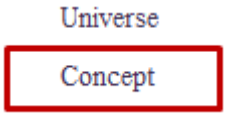

Figure 4: Concept

With the definitions of frame and concept it is entirely possible to design the scheme, see Fig. 5.

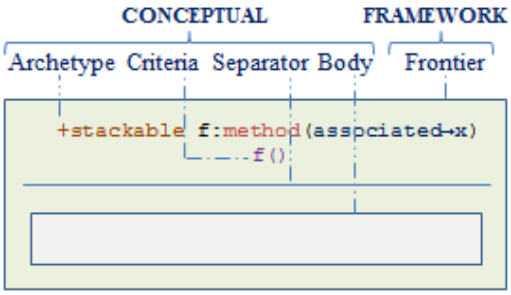

Figure 5: Concept Frame

Archetype: The archetype sets properties, identification, category and concept interaction with the universe of discourse. See Fig. 6.

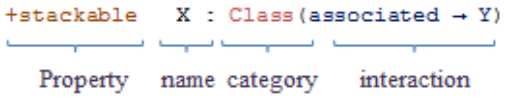

Figure 6: Archetype

The properties allow security features, storage and ways to change the concept to be set. A concept can be anonymous or have a name in which case allows a peculiarity or "instance" to be established. The category defines whether the concept is in the domain of the object language or the domain of the meta-language; with the interaction, archetype allows relationships, connections and links to be explicitly established.

Criteria: Set the concept, establishing its definition, interaction, and possible ways of inquiry, testing, and processing.

Separator: defines the boundary between criterion and body, the body itself also separates a conceptual framework from another. The separator marks the criterion and also establishes how the body should be interpreted. There are two interpretations: mode and type. Mode determines whether the interpretation is parallel or sequential. Type determines whether the interpretation is direct or recursive.

Body: is the set of conceptual frameworks, which, at the same time is contained in a conceptual framework.

4.2. Sintax of CFL

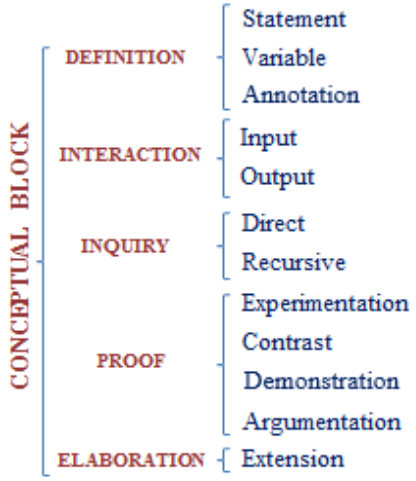

Figure 7: CFL Syntax

The syntax of CFL, Fig. 7, is based on the conceptual block, this consists of: definitions, inquiries, interactions, proofs and elaborations.

Definition: as in any programming language, there is a concept block in CFL in which three kinds of definitions can be made: variables, statement, and annotations. A variable definition is used to form the containers of information; a statement definition is used to propose invocations, returns and overall sentences in which the variables are used; finally, an annotation definition is used for documentation.

Interaction: with interactions CFL allows the user to manage input and output through which it is possible to communicate desired conditions for a program's execution.

Inquiry: CFL presents a model based on the formulation of questions about the state that variables can take, this kind of inquiry can be direct or recursive. An inquiry is direct when driving to take one path or another once, while the inquiry takes a recursive way or another a number of times.

Proof: proof allows scenarios to be defined where results may be different for the same conditions, due to uncontrolled changes that variables can take in a given time. This concept can be compiled as experimentation, contrast, demonstration, argumentation, etc.

Elaboration: elaboration allows extensions to be defined to extend the language with premises, operations and conclusions.

4.2 Semantics of CFL

The configuration of the conceptual frameworks consistently, constitutes the semantics of CFL. See Fig. 8. With the structure of conceptual frameworks it is possible create semantics, which representing the solution of a problem. A conceptual method forms a module [6], which, depending on the information exchange can be a "process" if it receives and produces information to the context, a "procedure" if it receives and produces information for the

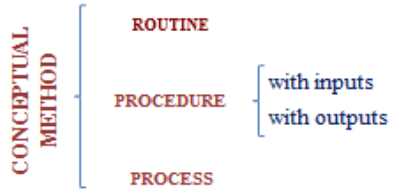

Figure 8: CFL Semantics

context or a "routine" if it does not receive or produce information for the context.

5. Schematic vs Diagram

CFL produces schemes, unlike diagrams, schematic left implicit relationships through the order and nesting of frames. Composition relations are simulated by the horizontal sequencing, inheritance and realization relations are assembled sequenced vertically between conceptual methods. If the method has a higher category it is located above, if has a subclass is located below. In the horizontal direction, the client is located on the left while the provider on the right. In the vertical a white box is used, while in the horizontal a black box is used [7]. See Fig. 9.

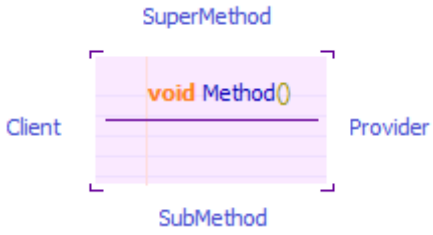

Figure 9: Implicit relations for a block

5.1. Color codes in CFL

In schemes, a color code is used, which enhances its meaning. Green was assigned to Definition, which symbolizes confidence, tranquility and development [8]. See Fig. 10.

Figure 10: Definition color (Green)

For Interaction orange was assigned, which symbolizes the striking, socialization and transformation [8]. See Fig. 11.

For Inquiry blue was assigned, it symbolizes science, idealism and functionalism [8]. See Fig. 12.

Figure 11: Interaction Color (Orange)

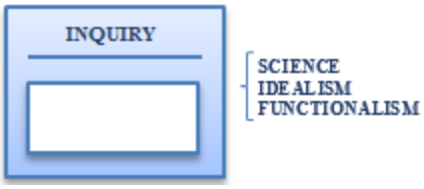

Figure 12: Inquiry Color (Blue)

For Elaboration red was proposed, which symbolizes prohibited, danger and dynamism [8]. See Fig. 13.

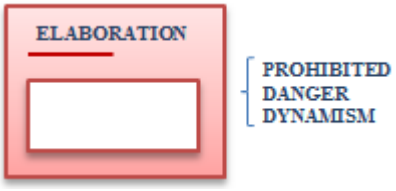

Figure 13: Elaboration color (Red)

For proof yellow was proposed, which symbolizes enlightenment, warning and creativity [8]. See Fig. 14.

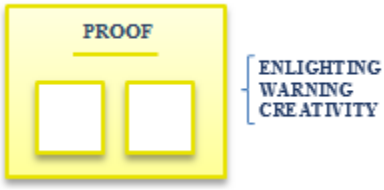

Figure 14: Proof color (Yellow)

6. CFL vs Other Languages

CFL allows for expression by similar classes as do object-oriented models, such a class in UML [9] and Java can be represented as Fig. 15.

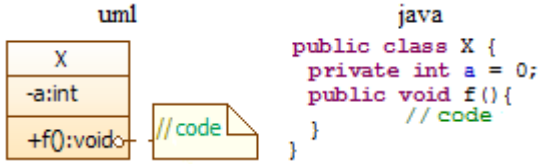

Figure 15: Class in UML and Java

CFL has the schematic, Fig. 16.

In the representation language, according to the productions in CFL:

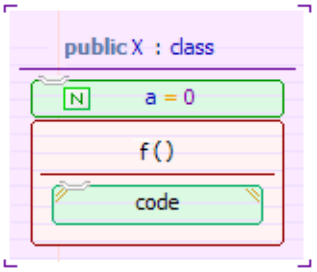

Figure 16: Category in CFL

```
{
  'public X:class' |
   [
    {[ private int a = 0 ]}
    { public void f() | [ {[code]} ]}
   ]
}
```

The schemes aim is to hide the formal layer, which is useful for language development and transparent to the user.

7. CFL Principles

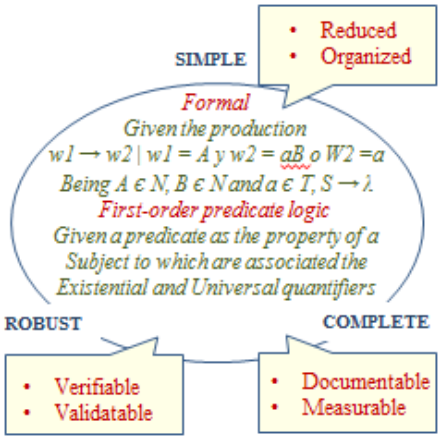

Figure 17: CFL Principles

CFL is simple, robust and complete, Fig. 17, these three principles underlie the language. Simplicity [10], reduces the work, thanks to a set of abstractions with the highest level of those used in a programming language, because only blocks are used, configured according to the desired model. Also, the organization of the blocks is automatic, reducing the learning curve, there is a marked difference with programming or modeling languages.

The CFL scheme synthesizes text and diagram; in both cases, the programming language and design, the complexity is reduced 11]. CFL is strong in dealing with concepts which allow the verification of algorithms directly and transparently through using the tracing of their errors, including validation of inputs and outputs. On the other hand, it is very complete, allowing documentation to be included using a native mechanism and providing direct metrics. CFL is formal due to the sound formation of its structure, which contains vocabulary and well defined production rules.

8. CFL Metrics

The metrics in software allows development control. In CFL two metrics are proposed, uncertainty balance and algorithm density [12].

8.1. Uncertainty balance

Software development is a creative exercise, which begins with a problem domain with great uncertainty to reach a solution domain with high certainty. The exercise of developing software is to eliminate uncertainty gradually to approach the certainty in which the solution is given.

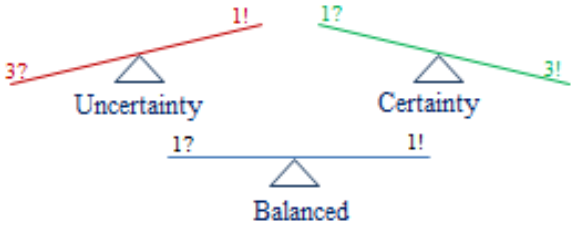

Figure 18: Balance of uncertainty

This metric lists and shows visually inquiry frames versus definition frames, in order to assess the degree of certainty that will be achieved in developing a solution, and constitutes an important source of choice for the design of a conceptual method. This metric produces three scenarios, see Fig. 18.

In the first type of balance there are more questions than answers, producing high uncertainty, and this can cause difficulty in the developments. The type of balance where questions match responses is the type "balanced", this is adequate to address the uncertainty, as each question has its solution. The third type of balance is more important in responses, usually due to responses that are part of protocols. The trend in software is the "balanced" type, because it has a one to one correspondence between the problem and the solution.

8.2. Algorithmic density

This type of metric is similar in information to the uncertainty balance. The difference is just the graphic representation, which seeks to represent inquiries and definitions as areas, which should have a tendency to follow a Gaussian bell, see Fig. 19.

Figure 19: Algorithmic density

9. CFL Platform

The Coloso platform for CFL is an application developed in the Java language, using the SWT framework [13]. See Fig. 20.

The framework is managed with dialogues that summarize the semantic possibilities of CFL, this first layer of interaction with the user sets the first filter of CFL expressions. The second filter is set once the conceptual method is run in the background which launches an application on the fly that is loaded and compiled in Java, the result is uploaded and presented in the first layer without the user having to know the details of the base language.

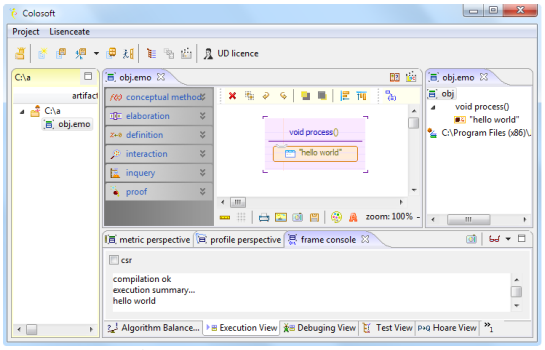

Figure 20: Coloso Software.

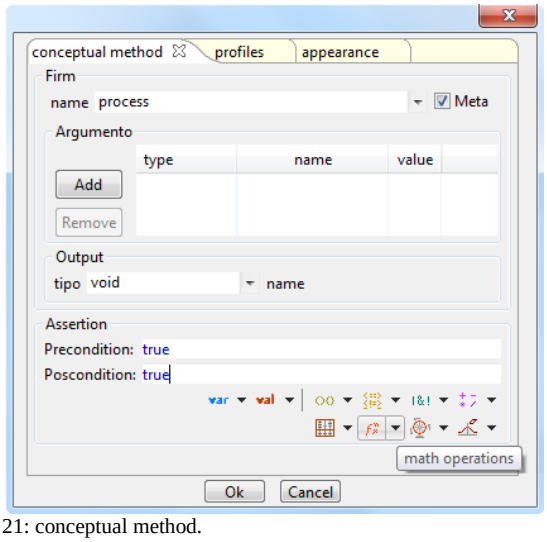

Figure 21: conceptual method.

Conceptual method, Figure 21, can manage inputs and outputs, pre and post conditions, also, all conceptual framework incorporates a toolbar in which the variables that should be used when using a framework are found, also suggested values, operators, groupers and native language operations are presented.

Besides being executed, a framework can be debugged, validated, verified, and measured, the outputs can be evidenced in each of the views, dedicated for every approach.

Debugging allows tracking step by step a conceptual method, presenting the state of the variables defined in the framework. Verification and validation of conceptual methods may be evident in their view, the verification is displayed as a tree that maps errors, faults and defects. In the validation perspective a tabulated Hoare triplet [14-15] highlights the corresponding evaluation method. CFL presents a view of metrics in which all the framesets used are listed and allows, along with the view of density algorithms and algorithmic balance, one to see the trend of the conceptual method, supported by recommendations based on cyclomatic complexity [16], and a magic number 7 ± 2 [17].

CFL also provides ease of integration with languages like UML, this bridge closes the gap between the class diagram and the programming language, specifically this facility allows the generation of an operation code

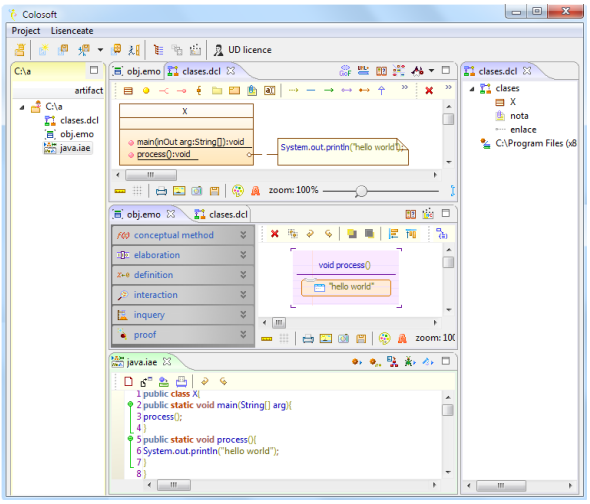

Figure 22: UML/CFL/Java integration

belonging to a class and the algorithm solved in a conceptual method and that is associated with that operation. See Fig. 22.

10. Case Studies

The following describes the tests performed to validate the principles of simplicity, robustness and completeness of CFL.

To perform the test of simplicity, we measured the time spent in developing a solution to a given problem, for that, the effort was compared, measured in time needed to solve the set of algorithms that conform a course of programming and algorithms. The test included 20 algorithms and was applied to teachers who teach the area. The course is normally conducted with tools like DFD [18], PSeInt [19] or programming languages like Java, the sample used includes problems like: traversals, exchanges, queries and sorts. Five problems were formulated for each subject and divided into two groups each with two teachers, one group chose the tool that they had been using before, "DFD", this group was called "alternate group", the second group chose CFL, this group was called "CFL group". Table 1 tabulates the time used in minutes, together with the obtained average μ and standard deviation σ, according to equations 2 and 3.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} a_i = \frac{a_1 + a_2 + \cdots + a_n}{n} \quad (2)$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(X_i - \mu)^2}{n}} \quad (3)$$

The average development time of the algorithms in the CFL group was much lower than the alternate group. The dispersion of the CFL sample was low while the alternate group was high, the fundamental reason was reflected, in particular, in a sorting algorithm: the problem of Hanoi

Table 1
Simplicity test

	Group	
Topic	Alternate	CFL
Exchanges	22	20
Traversal	30	23
Queries	40	27
Sorts	80	30
μ	**43**	**25**
σ	**22.29**	**3.80**

Table 2
Robustness test

	Group	
Topic	Alternate	CFL
Exchanges	5	5
Traversal	3	5
Queries	2	5
Sorts	1	5
μ	**2.75**	**5**
σ	**1.47**	**0**

Towers [20], this problem was easily solved by the CFL group, but the alternate group was limited in DFD to solve iteratively, this greatly increased the solution time.

The next test sought to establish the robustness, at this point became the possibility of using a programming language if desired. The test consisted of breaking algorithms with invalid entries in a total time of one hour. The table shows the number of algorithms tested (successfully) by category.

CFL provide great ease of use thanks to its direct and transparent management of assertion concept.

For the alternate group, which took the option of changing to Java, it became a time consuming task because the algorithms had to be migrated, as they couldn't use the assert concept and reduce the task just to do comparisons. In this test, the difference between the CFL group and the alternate group was emphasized.

The third test was conducted to test completeness conditions, for this, each group was requested to explain the model or code that solved each problem, using some metric about the algorithms. The time for this activity was one hour. Table 3 presents the number of documented algorithms.

All algorithms for both groups were documented, but the documentation of alternate group was made apart, this decision was taken because two different tools were used, so, two separate documents should be created, otherwise, it would be necessary to make the documentation by using a third tool. Also, this documentation was not enriched with metric criteria. On the other hand the CFL group documentation was integrated into the program, based on the proposed metric for the language.

The above tests give a favorable starting point to CFL, however, for future work we propose to perform more extensive testing.

Table 3
Completeness test

	Group	
Topic	Alternate	CFL
Exchanges	5	5
Traversal	5	5
Queries	5	5
Sorts	5	5
μ	**5**	**5**
σ	**0**	**0**

11. Conclusions

Alpha testing performed on CFL, shows how the language facilitates the resolution of computational problems, due to the abstraction degree that it provides, hiding the complexity associated with knowledge of semantics and syntax of a computer language.

The schemes used by CFL, as a graphic representation, eliminates the complexity introduced by the relationships of a conventional graphical model, instead, it uses assembly going to the mental model of sequence and nesting assembly sequence model and nesting.

CFL emphasizes problem solving by using a scientific vocabulary, in which the predominant characteristics are the inquiry (observation), the definition of variables, the definition of tests and experiments, and elaborations; computational features characteristic of programming languages, are in the background.

References

[1] Budd, T. Programación Orientada a Objetos. Addison-Wesley, 1994.

[2] Budge, E. A. Wallis E., Sir. Rosetta stone. London: British Museum. 1857-1934.

[3] Tucker, A., Noonan, R. Programming Languages Principles and Paradigms. McGraw Hill, 2002.

[4] Rosen, K. H. Matemática Discreta. Mc Graw Hill, 2004.

[5] Teufel, B., Schimidt, S. T. Compiladores conceptos fundamentales. Addison-Wesley, 1995.

[6] Meyer, B. Construcción de Software Orientado a Objetos. Prentice Hall, 1999.

[7] Gamma, E., Helm, R., Johnson, R., & Vlisides, J. Design Patterns. Addison-Wesley, 1995.

[8] Heller, E. Psicología del Color. Editorial Gustavo Gili, 2007.

[9] Booch, G., Rumbaugh, J., & Jacobson, I. The Unified Modeling Language User Guide. Addison-Wesley, 2005.

[10] Maeda, J. Las Leyes de la Simplicidad. Gedisa, S.A., 2005.

[11] Morin, E. Introducción al Pensamiento Complejo. Gedisa, S.A., 2005.

[12] Bolaños, S., Medina, V., & Aguilar, J. Principios para la Fromalización de la Ingeniería de Software. Ingeniería, pp. 31-37, 2009.

[13] Hatton, R. Swt: A Developer's Notebook. O'Reilly & Associates, 2005.

[14] Hoare, C.A.R. An Axiomatic Basis for Computer Programming. Communication of the ACM. ACM. Vol 12. No 10, 1969.

[15] Hoare, C.A.R. Viewpoint. Retrospective: an axiomatic basis for computer programming. Communication of the ACM. ACM. Vol 52. No 10, 2009.

[16] Mccabe, T. A Complexity Measure. IEEE Transactions on Software Engineering. IEEE Journal & Magazines Vol SE-2 No 4. Pp 333-349, 1976.

[17] Miller, G. A. The magical number seven, plus or minus two: Some limits on our capacity for processing information, Psychological Review. Vol. 63. No 2. pp. 81–97, 1956.

[18] Cárdenas, F. Daza, E. Castillo, N. 1996. DFD. Online: http://freedfd.googlecode.com/files/FreeDFD-1.1.zip, 2012

[19] PSeInt. Online: http://pseint.sourceforge.net/, 2012

[20] Pickover, C. The Math Book. Sterling Publishing Co. Inc., 2011.

S. J. Bolaños Castro, PhD from the Pontifical University of Salamanca in Madrid, Spain and PhD Prize awarded by the same university; graduated as a systems engineer and teleinformatics magister in the University Francisco José de Caldas in Bogotá, Colombia. He is a member of the international research group GICOGE; his researcher interest is about Software Engineering. He is teacher about Software Engineering in several programs in the university. He is also director of curriculum projects of undergraduate and graduate, currently heads the graduate in Software Engineering and IT projects at the University Francisco José de Caldas District.

R. González Crespo, is the deputy director of the School of Engineering at the Universidad Internacional de La Rioja. Professor of Project Management and Engineering of Web sites. He is honorary professor and guest of various institutions such as the University of Oviedo and University Francisco José de Caldas. Previously, he worked as Manager and Director of Graduate Chair in the School of Engineering and Architecture at the Pontifical University of Salamanca for over 10 years. He has participated in numerous projects I + D + I such as SEACW, GMOSS, eInkPlusPlusy among others. He advises a number of public and private, national and international institutions. His research and scientific production focuses on accessibility, web engineering, mobile technologies and project management. He has published more than 80 works in indexed research journals, books, book chapters and conferences.

V. H. Medina García, PhD in Computer Engineering from the Pontifical University of Salamanca, Master in Computer Science from the Polytechnic University of Madrid, Specialist in Marketing from the University of Rosario, Systems Engineer from the District University Francisco José de Caldas; Director of the PhD. program in Engineering from the District University Francisco José de Caldas District in Bogota Colombia. Writer, speaker and teacher recognized internationally; he has published several books and conducted various academic and administrative units. He is academic and principal investigator of the research group GICOGE, category A.1 in COLCIENCIAS.

J. Barón Velandia, MsC. In teleinformatics, systems engineer, and software engineering teacher at District University "Francisco Jose de Caldas" in Bogotá, Colombia; PhD student at the Pontifical University of Salamanca in Madrid, Spain. INTECSE investigation group's founder and director. Former REDIS director and System Engineering undergraduate's coordinator at District University.