DE GRUYTER OPEN

DOI: 10.1515/aucts-2016-0007

ACTA UIVERSITATIS CIBINIENSIS – TECHNICAL SERIES
Vol. LXVIII 2016

A LINGUISTIC MODEL IN COMPONENT ORIENTED PROGRAMMING

Crăciunean Daniel Cristian
(Ph.D. student), Faculty of Engineering/Department of Computers and Information Technology, "Lucian Blaga" University of Sibiu, Sibiu, Romania, daniel.craciunean@gmail.com

Crăciunean Vasile
(associate professor / Ph.D.), Faculty of Engineering/Department of Computers and Information Technology, "Lucian Blaga" University of Sibiu, Sibiu, Romania, vasile.craciunean@ulbsibiu.ro

Abstract: *It is a fact that the component-oriented programming, well organized, can bring a large increase in efficiency in the development of large software systems. This paper proposes a model for building software systems by assembling components that can operate independently of each other. The model is based on a computing environment that runs parallel and distributed applications. This paper introduces concepts as: abstract aggregation scheme and aggregation application. Basically, an aggregation application is an application that is obtained by combining corresponding components. In our model an aggregation application is a word in a language.*

Keywords - abstract aggregation scheme, aggregation component, aggregation computing block, aggregated application, network of aggregated applications.

1. Introduction

Component-oriented programming is a living hope at the moment in the world of software developers in terms of increasing efficiency. There is a general consensus that the most effective reuse of written code, are components that can function in a suitable environment independently from the other components. A component includes in it all the necessary information to be used by a client.

Component-oriented programming involves two distinct activities [2]: create new components and creating applications by aggregating existing components. Therefore when we want to develop a component oriented application will have to first try to build on existing components and just after that to build new components needed for our application requirement. This components will enlarge our base of components.

Our base of components, on which we build applications, together with facilities for component construction and management is the component infrastructure.

The component infrastructure consists of three distinct models: a component model, a component connection model, and a component deployment model.

A component model defines what a valid component is, and how to create a new component in the component infrastructure. All reusable components will be built, thus, according to the model of the component. Each component infrastructure has a library of reusable components conforming to the component model.

The component connection model defines a collection of connectors and support facilities for component aggregation. Therefore, the connection model determines how to build an application or a larger component from existing components.

The component deployment model describes how the components will be implemented in a work environment.

The component infrastructure is also called component-oriented technology or component-oriented architecture.

Based on this reality and the belief that this is the evolution of software development, this paper proposes a linguistic model for building complex application starting from components. Simplifying things, in this paper an application is a word in a language. All the components of an application are designed to operate in a parallel and distributed environment.

In 3. we introduce the notion of an abstract aggregation scheme (S) and the language L(S) associated with it. The abstract aggregation scheme is a general framework that defines an aggregated application. In 4. we introduce the notion of network of aggregated applications and functional aggregated application.

We believe that the present paper provides the necessary mechanisms to build static or dynamic optimal applications by assembling independent components.

2. Preliminary Notions and Notations

Let X be a set. The family of subsets of X is denoted by $\mathcal{P}(X)$. The cardinality of X is denoted by |X|. The set of natural numbers, {0, 1, 2, ...} is denoted by N. The empty set is denoted by Ø. An *alphabet* is a finite nonempty set of abstract symbols. For an alphabet V we denote by V^* the set of all strings of symbols in V. The empty string is denoted by λ. The set of nonempty strings over V, that is $V^*-\{\lambda\}$, is denoted by V^+. Let Sub(w) denote the set of subwords of w. Each subset of V^* is called a language over V. The *length* of a string $x \in V^*$ (the number of symbol occurrences in X) is denoted by |x|. The number of occurrences of a given symbol $a \in V$ in $x \in V^*$ is denoted by $|x|_a$. Let Symb(w) denote the set of symbol occurrences in w.

3. Abstract Aggregation Scheme

Definition 1. An abstract aggregation scheme is a construct of the form

$S = (A, \rho, f, \{L_X \mid X \in \mathcal{P}(A_\rho)\})$

where

A is a finite set of elements called elementary components;

ρ is an equivalence relation over A that we name a cohabitation relation over A.

We denote by $A_\rho = A|_\rho$ the set of equivalence classes of A with respect to this relation. The elements of A_ρ are called components;

f is a function $f: A_\rho \rightarrow \mathcal{P}(A_\rho)$;

L_X is a language over A_ρ, $(\forall)\ X \in \mathcal{P}(A_\rho)$.

An elementary component $c \in A$ performs a single action. The aggregation components, which can contain multiple actions, will be formed by grouping several elementary components that are equivalent relative to the cohabitation relationship. The cohabitation relationship ρ solves a problem that is quite common, where certain elementary components, for various reasons, can only be used in the presence of other elementary components.

If $A_\rho \neq A$ then we say that S is an abstract aggregation scheme with cohabitation. Otherwise, it a non-cohabitation abstract aggregation scheme.

Starting from f we now define a function φ as follows:

$\varphi: \mathcal{P}(A_\rho) \rightarrow \mathcal{P}(A_\rho)$

$$\varphi(X) = \bigcup_{x \in X} f(x) \quad (\forall)\ X \in \mathcal{P}(A_\rho)$$

it's clear that $\varphi^n(X)$ can be defined as

$\varphi^0(X) = X \quad (\forall)\ X \in \mathcal{P}(A_\rho)$

$\varphi^n(X) = \varphi(\varphi^{n-1}(X))$

Similarly we use the notation:

$\varphi^{-1}(X) = \{x \mid f(x) \cap X \neq \emptyset\}$

$\varphi^{-n}(X) = \varphi^{-1}(\varphi^{-(n-1)}(X))$

Also we observe that $\varphi^n(X) \in \mathcal{P}(A_\rho)$ $(\forall)\ n \in N$ and $(\forall)\ X \in \mathcal{P}(A_\rho)$, and therefore over the set $\varphi^n(X)$ we have the language $L_{\varphi^n(X)}$ as defined in *Definition 1*.

The set $\{\varphi^n(X);\ n \geq 0,\ X \in \mathcal{P}(A_\rho)\}$ is finite, because A_ρ is a finite set.

Definition 2. Let

$S = (A, \rho, f, \{L_X \mid X \in \mathcal{P}(A_\rho)\})$

be an abstract aggregation scheme. Then the language L(S) specified by the abstract aggregation scheme S is:

$$L(S) = \bigcup_{X \in \mathcal{P}(A_\rho)} \bigcup_{i=0}^{\infty} \prod_{k=0}^{i} L_{\varphi^k(X)}$$

Lemma 1. Let

$S = (A, \rho, f, \{L_X \mid X \in \mathcal{P}(A_\rho)\})$

be an abstract aggregation scheme and the language

$$R(X) = \bigcup_{i=0}^{\infty} \prod_{k=0}^{i} L_{\varphi^k(X)}$$

then

$R(X) = R_1(X) \cup R_2(X)(R_3(X))^* R_4(X)$

where

$$R_1(X) = \bigcup_{i=0}^{i_1-1} \prod_{k=0}^{i} L_{\varphi^k(X)} \qquad R_2(X) = \prod_{k=0}^{i_1-1} L_{\varphi^k(X)}$$

$$R_3(X) = \prod_{k=i_1}^{i_2-1} L_{\varphi^k(X)} \qquad R_4(X) = \bigcup_{i=i_1}^{i_2-2} \prod_{k=i_1}^{i} L_{\varphi^k(X)}$$

Proof. For $X \in \mathcal{P}(A_\rho)$ we consider the following sequence:

$X, \varphi(X), \varphi^2(X), \ldots, \varphi^k(X), \ldots$

Because the set $X \subset A_\rho$ is finite, it's obvious that $(\exists)$ i and j such that $\varphi^i(X) = \varphi^j(X)$ and therefore $L_{\varphi^i(X)} = L_{\varphi^j(X)}$.

Let i_1 and i_2 ($i_1 < i_2$) be the smallest such values for i and j. If we denote

$$R_1(X) = \bigcup_{i=0}^{i_1-1} \prod_{k=0}^{i} L_{\varphi^k(X)} \qquad R_2(X) = \prod_{k=0}^{i_1-1} L_{\varphi^k(X)}$$

$$R_3(X) = \prod_{k=i_1}^{i_2-1} L_{\varphi^k(X)} \qquad R_4(X) = \bigcup_{i=i_1}^{i_2-2} \prod_{k=i_1}^{i} L_{\varphi^k(X)}$$

then the Lemma 1 is proved.

We have a prefix from 0 to i_1 which is not repeated, followed by R_3 which can repeat from 0 to infinite and then we have a remainder (a part from a period).

Theorem 1. Let

$S = (A, \rho, f, \{L_X \mid X \in \mathcal{P}(A_\rho)\})$

be an abstract aggregation scheme. If all languages L_X, $X \in \mathcal{P}(A_\rho)$ are of type $i \in \{0,1,2,3\}$ in the Chomsky hierarchy, then the language L(S) is of type i in the Chomsky hierarchy[1][3][4].

Proof. It follows immediately from closing the languages from the Chomsky hierarchy at union, catenation, catenation closure and from Lemma 1.

4. Aggregation Scheme of the Components

We concretize now the notion of elementary component and component.

Definition 3. An elementary component $c \in A$ is a construct of the form

$c = (i, o, m)$

where

$i=(d_i:T_i, e_i:U_i)$ is a pair (parameter, event) which forms the entrance to the elementary component, d_i and e_i are identifiers and T_i and U_i are the types of these identifiers;

$d_i:T_i$ is the input parameter;

$e_i:U_i$ is the input event that triggers action m.

$o=(d_o:T_o, e_o:U_o)$ is a pair (parameter, event) which forms the output from the elementary component, d_o and e_o are identifiers and T_o and U_o are the types of these identifiers;

$d_o:T_o$ is the output parameter of the elementary component;

$e_o:U_o$ is the output event of the elementary component.

m is the appropriate action of the elementary component. This is the action that the elementary component performs when it receives the U_i event and the input parameter $d_i:T_i$.

We will further consider an elementary component as a computational process which admits an object and an event with the types specified by i as input and an object and an event with the types specified by o as output.

Input objects, events and actions are considered to be standardized, i.e. any element that performs the same action will have the same input, same output and the same action. Thus the elementary components may be invoked by actions, input parameters and events without knowing their names.

Definition 4. A aggregation component *a* is a construct of the form

a = (I,O,M,u)

where

$I=\{i_c \mid c \in a\}$;

$O=\{o_c \mid c \in a\}$;

$M=\{m_c \mid c \in a\}$;

$u: I \rightarrow M$ is a function called an action selection function.

In this conditions we will define the function f: $A_\rho \rightarrow \mathcal{P}(A_\rho)$, from the definition of the abstract aggregation scheme, like this:

$f(a) = \{b \mid O_a \cap I_b \neq \emptyset\}, (\forall)\ a \in A_\rho$

Let S be an abstract aggregation scheme with the aggregation components defined as in *Definition 5*. Then every word $\alpha \in L_{\varphi^k(X)}$; $X \epsilon\ \mathcal{P}(A_\rho)$ defines an aggregation computing block.

If we have a string of aggregation components $\alpha=a_1a_2...a_m$ where $a \in Symb(\alpha)$ is of the form

$a = (I_a, O_a, M_a)$ then we denote

$$I_\alpha = \bigcup_{a \in Symb(\alpha)} I_a$$

and with

$$O_\alpha = \bigcup_{a \in Symb(\alpha)} O_a$$

Definition 6. An aggregation computing block is a construct of the form

$\mathcal{B} = (I, O, \alpha, B)$

where $\alpha \in L_{\varphi^k(X)}$; $X \epsilon\ \mathcal{P}(A_\rho)$; $I=I_\alpha$; $O=O_\alpha$;

B is an object, named block-board, that can store information of the form (parameter, event). B is used by the aggregation components for communication.

Definition 7. Let

$S = (A, \rho, f, \{L_X \mid X \in \mathcal{P}(A_\rho)\})$

be an abstract aggregation scheme and $X \in \mathcal{P}(A_\rho)$. Then the language

$$R(X) = \bigcup_{i=0}^{\infty} \prod_{k=0}^{i} L_{\varphi^k(X)}$$

is called an aggregation network generated by X.

Definition 8. Let

$S = (A, \rho, f, \{L_X \mid X \in \mathcal{P}(A_\rho)\})$

be an abstract aggregation scheme, R(X) an aggregation network generated by X and $\alpha=\alpha_1\ \alpha_2\ \dots\ \alpha_n$, $\alpha_i \in L_{\varphi^i(X)}$ a finite word from R_X. Therefore, $\mathcal{A}$= (I, O, α,B) is called an aggregate application generated by X.

$$I = I_{\alpha_1} \qquad O = \bigcup_{\alpha_i} O_{\alpha_i}$$

B is an object, called application-board, that can store information of the form (parameter, event). B is used by the aggregation components for communication.

Definition 9. Let S be an abstract aggregation scheme and $\mathcal{A}$= (I,O,α,B) an aggregate application generated by X and $\alpha=\alpha_1\ \alpha_2\ \dots\ \alpha_n$, $\alpha_i \in L_{\varphi^i(X)}$.Then the application $\mathcal{A}$ is a functional aggregate application generated by X if (∀) i=1, …, n-1 we have $I_{\alpha_{i+1}} \subseteq O_{\alpha_i}$.

Definition 10. Let

$S = (A, \rho, f, \{L_X \mid X \in \mathcal{P}(A_\rho)\})$

be an abstract aggregation scheme. Then

$$R(X,n) = \prod_{k=0}^{n} L_{\varphi^k(X)}\ , n \in N$$

is called a network of aggregate applications generated by X and length n.

We remark that R(X, n) contains all aggregate applications generated by X and length n, namely all aggregate applications of the form:
$\mathcal{A}$= (I,O,α,B)

$$I = \bigcup_{a \in X} I_a \qquad O = \bigcup_{k=0}^{n} \bigcup_{a \in \varphi^k(X)} O_a$$

α∈R(X,n) , (B_i, i=1, …, n) and B are like in the *Definition 7*.

In practice we are interested in an application that has a lot of inputs I and plenty of outputs O. Obviously, we need to search for our application in a subset of R(X, n), which we denote R(X, Y, n) where:
X is a minimal set Z with the property $\bigcup_{a \in Z} I_a \supseteq I$

$$n = \min\left\{ i | O \subset \bigcup_{k=0}^{i} \bigcup_{a \in \varphi^k(X)} O_a \right\}$$

Y = {a∈φ^n(X) | O_a∩O≠Ø}

$$R(X,Y,n) = \prod_{k=0}^{n} L_{(\varphi^k(X) \cap \varphi^{-(n-k)}(y))} \ , n \in N$$

Definition 11. The language R(X, Y, n) is called an acceptable network of aggregate applications.

5. Conclusions

The purpose of this model is to generate applications by combining components. An application $\mathcal{A}$ defined as in *Definition 8* will operate in parallel and distributed environment. Starting the system involves parallel activation of all aggregation components. The elementary components of the same component will communicate with each other by means of a block-board associated with the corresponding block. The access to block-board is synchronized.

Each component will get its input data from the application-board and write all its output data also on the application-board. Of course, these operations will be synchronized using semaphores associated with each type from the application-board.

We notice that we can get many applications that do the same thing from the user perspective, namely the applications have the same input and output.

It's natural that we put the question to choose the optimal application to solve a particular problem, optimal in terms of execution time and resources. The problem to find the optimal application can be put in two ways:

i) Static determination of an optimal application.
ii) Building an application that evolves in time, changing its components in concordance of their efficiency.

6. References

1. Aho, Alfred V., Sethi Ravi, Ullman Jeffrey D., *Compilers: Principles, Techniques, and Tools*, Addison Wesley, (2001).
2. Wang, Andy Ju An., Qian, Kai., *Component-Oriented Programming* , John Wiley & Sons, (2005).
3. Salomaa, A., *Formal Languages*, New York, Academic Press, (1973).
4. Păun Gh., *Probleme actuale in teoria limbajelor formale*, Editura Academiei, (1983).