From Competition to Amalgamation of Different Programming Paradigms

Sergei G. Maslov

Udmurt University Dept. of Mathematics & Computer Science Russia

p.c.: 426011, city: Izhevsk, 275-61 Udmurtskay str. e-mail: miss@matsim.udmurtia.su, Fax: 3412 75 15 18 NATO ASI Constraint Programming August 13-24 93 Pärnu, Estonia

Abstract

This paper describes the basic elements (data, actions, tunings, voids, mixtures,...) and principles of programming (stratification, implicit knowledge, limited freedom,...), directed to amalgamating of different programming paradigms (imperative, object-oriented, functional, constraint,...) in a unified process of generating computer system models. Compositions of elements and their projections on axes of representation (visual, audio, tactile, and linguistic) make it possible to construct different algorithmic structures, which have their own syntactic form and operating semantics. Development of compositions is a goal-oriented activity controlled by constraints (resource, domain, coexistence,...). This action creates both stratification system of concepts and operations on them (generalization, specialization, mapping, transformation,...). A proposed approach is principally evolutionary one. It is devoted to clarify the fundamental basis of forms and principles of a heterogeneous knowledge represented in a computer. It allows to stratify program synthesis into levels of representation. In particular, some aspects of conceptual synthesis are concerned.

1. Introduction

The research and development process is a movement on the boundary between the known and the unknown. It is characterized by subjective perception of the object in view. Both require repeated and prompt modification of the problem statement and advanced versatile interface, helping to detect additional analogies and to define more exactly "blank spaces" in the research object. Incomplete knowledge causes irregularity of the research process, frequent moves

from one aspect of the research object to another, or even simultaneously investigating them all. The semantic layers are activated or avoided in accordance with a research goal and with the current computation state and the subject's actions.

Enormous volume and relational complexity of knowledge in computer system models (CSM), heterogeneity of the knowledge and representation forms result in the incapability for a single or more designers from one branch of knowledge to define faithfully and to research and develop an object. Here versatile, coordinated and unique definitions are most important factors.

The most acute problems are adequacy of the description means, the means to organize a professional's knowledge interaction in a computer form, problems of search and recognition. Solving of these problems allows to move from the collection of facts and processes (database or knowledge base) to *knowledge investment*.

Obviously, in this situation something more is required than what is involved in each particular programming paradigm. Simple replacement of the programming paradigm doesn't help either, but can worsen the situation, leading to endless programming and provoking a revolutanary way of designing.

One of the current approaches to solving problems that arise is amalgamating of programming paradigms (imperative, object-oriented, functional, logic, concurrent, constraint) in a unified process of CSM construction. This approach is known and used in the following forms:

- procedural extention of programming language (Lisp [HS86], Forth [TF]);
- syntax and semantic design and development of languages (Kaleidoscope [LFB93], AKL [HJ93], Life [AKP93], FP + LV [DGK93], Falcon [GY93], NUT [MT92, PT93])
- research of different formalizations (Horn clause logic with equality [GM87], Linear logic [DGK93], OSF logic & algebra [AKP93], ...).

Here a constructive base of this approarch is extended and systematized. It is formed on the basis of stratification, taking into account the heterogenity of knowledge forms in CSM. What is a basic set of constructive elements? What properties must they have to represent clearly the compound nature of the object being researched and to flexibly control its construction? What is the crystallizing compound that unites the elements in the composition? Where is that boundary or the moment of crossover from one programming paradigm to another or their amalgamating? What economizes our intellectual, physical and emotional resources and strictly directs to the stated goal? These questions were emerging regularly to the author during the creation of a CSM for researching the goal-directed three-dimensional motions of a human.

2. About elements

Development of a conceptual system for special reseach area is accompanied by regularly including new concepts, the revision of parameterization, the base types of data and algorithm representation, the different representation forms (linguistic, visual, audio, tactile, ...), the continual improvement of hardware - all of which, from one hand, permit us to represent a research object more effectively, adequately and from different viewpoints, and from the other hand, require constant programming and reprogramming. However the enormous volume and complex structures of algorithms and data make reprogramming undesirable, i. e. minimalization or only automatization is needed.

Experience shows that it is insufficient for complex domains to construct simply semantically stratified descriptions: what is needed is to declare in them the *fundemental basis, that is genotype and phenotype* with the contruction of a transformation system (first with the operations of identification, generalization, specialization, imaging, and transformation). Such systematization, as a rule, has a *principally evolutional character* and will probably be accompanied by the collapse of semantic layers. Then the internal "life" of CSM with its own laws of existence and development arises.

Concepts are characterized by a certain set of degrees of freedom, which gives a possibility to compose of them. *Degrees of freedom* are the axes of multi-aspect or multi-functional representation of a concept. They represent, in particular, the form of representation, parameterization, the types of conversion, and semantic invariability relative to composition. Involving all the above, *the structure of concepts* can be represented as follows:

Concept

• Name • Features • Values • Relations • Parameters

Features

- Classes
- Properties

Values

- Evaluations
- Forms of Representation
 - Linguistic
 - Epistemological
 - Conceptual
 - Logical
 - Math

 - Algorithmic
 - Program
 - Video
 - Audio

Relations

- Operators
- Functions
- Equations
- Constraints
- Resources
 - Intellectual
 - Interfectu
 - Physical
 - Time
 - Memory
 - Economic
 - Emotional
 - Area
 - Domain
 - Coexistence

٠	Sensoric	• Structure
		\bullet Control

A more detailed representation can be found in article [MS93].

System and logical analysis of CSM development and use let us to clarify the basis B. of elements of constructive representation and to freeze a certain functional assignment:

- B_1 organize information in complex structure (mixture),
- B_2 generate information (action),
- B_3 coordination of information (tuning),
- B_4 preserving information (data),
- B_5 indeterminate information (void).

This classification was obtained independently from M. Lehman's SPE - classification [LM80], but corrobarate his major conclusion on the evolution of programs. Moreover this classification is brought to a concrete form of representation [MS93].

The element of a class is called a concept. Any element of a conceptual system is constructed either by defining a concrete form of representation, or by composing concepts from some concept set.

3. About compositions

The definition and usage associate concepts to the algorithmic structure (AS). This structure has its own syntax form and mechanism of calculation.

It is necessary to list a number of features.

	•[Ex/Im]plicit	•Parametrisability
Facture of the Suntar Form	•Richness of Associations	•Causality
reature of the Syntax Form	\bullet Separation	$\bullet Computability$
	•Invariance	

The feature [ex/im]plicit is connected to constructivity and descriptivity of syntax form. Parametrizability is the possibility to translate quite flexy the requested syntax unit into formal or actual parameter of syntax form. Richness of Associations is the restriction of syntax forms, which is made by means of direct references or special or special algorithms. Causality is ability of syntax forms or its units to activate and synchronize its roles. In other words, the states of compute process can activate various interpretations of AS and concepts involved in AS (as processes, data or parameters of operating of compute process). Separation express the spliting of syntax form in accordance with access mode, scope, and semantic independence. Computability is a possibility of computing of syntax form and reseiving various types of result (numeric, logic, symbol, image,...). It is possible to design new features of syntax form using the base features, for example:

Inheritance • Richness of Associations • Causality

The features are considered as degrees of freedom in construction of syntax form.

The creation of compositions is the goal-oriented process of building of new concepts with given properties. This action is accompanied with utilization of being allocated resources and imposing constraints on a set of concepts. Actually, we impose linkages on degrees of freedom of syntax units. Coordination of roles of various concepts in the AS is in the same time designing definition units and interpretation rules in calculation process.

Rules of Compositions

w

Concept	Data	Action	Data	Action
Data	Structure	Object	,	
Action	Function	Construct		
Data			Lazy comp.	Partial comp.
Action			Operated comp.	Nested comp.

A number of base operations can be extracted in process of construction of AS and operations with it:

• Generalization • Specialization • Mapping • Transformation

Generalization is the AS constructing from given set of AS by its uniting with possible truncation of individual features and matching of common ones. Specialization is constriction (or projection) of AS by given parameters, conditions or propeties. Transformation is AS mapping into another AS inside the range of given basis of techniques. Mapping is setting of correspondence between AS, in particular, from various layers of representation. More complex operations on AS can be constructed on base of above-described operations:

• Editing • Synthesis • Optimization • Analysis • Analitic conversion

Described system of concept is particular implemented as a system of microlanguages without low level (mx - algorithmic hypertext, mi - tensor & index notations, module(x) - module language...). These languages have unify syntax skeleton:

• glo	bal desc	cription $\square N \parallel C_1 \ldots$	$\ C_n \blacksquare$
• local description		iption $[N \parallel C_1 \dots$	$\parallel C_n$]
• mo	odificatio	on of description $[N \parallel C_1 \dots]$	$\ C_n \ $
● cal	1	$\begin{bmatrix} N P_1 & \dots & P_n \end{bmatrix}$]
	N	- name of notion	
	C	$\equiv A \Rightarrow B \Leftarrow D \Leftrightarrow S$	
hana	A, D	- constraints for B	
nere	B	- actived notion	
	S	- solution specification of B	
	P	- actual parameters	

Let us to illustrate the application of mx and mi languages.

Example - mx

```
\begin{array}{ccc} \mathbf{G} \text{ Greatest} \\ \parallel & \mathbf{X} > \mathbf{Y} \Rightarrow X \\ \parallel & \mathbf{X} \leq \mathbf{Y} \Rightarrow Y \end{array}
```

Definition of the three-dimensional position of the bodies composing a biomechanical system with a tree structure is reduced to construction of recurrent forms which are analitic expressions of the coordinates relative through generalized and other coordinates. Directed exhaustion of these forms is performed using a structure of the biomechanical system. For spherical joints it can be as follows:

Example - mi

```
 \begin{array}{l} & \text{Kinematics} \\ \parallel \psi_{..}\& \ z_{..}\& \ D_{..}\& \ ?x_{..} \Rightarrow \\ & [A \parallel (x_{\cdot p} + a_{\cdot i p} z_{\alpha i < \cdot >})_{\theta} = z_{\alpha p < 0 >} \\ & [ \ i \ p \parallel \ 1 \ 2 \ 3 \ ] \ \left[ \alpha \parallel \mid D_{\gamma 1} \mid \right] \quad \left[ \theta \parallel \ D_{\gamma 3} \right] \quad \left[ \gamma \parallel \ 1 \oplus \gamma + 1 \Leftarrow D_{\gamma 2} = 0 \right] \\ & \$ \gamma \{\alpha, \theta \{p\}\} ] \\ & [B \parallel (x_{\cdot p} + a_{\cdot i p} z_{\alpha i < >})_{\theta} = (x_{\cdot p} + a_{\cdot i p} z_{\alpha i < >})_{\theta} \\ & [ \ i \ p \parallel \ 1 \ 2 \ 3 \ ] \quad \left[ \alpha \parallel \mid D_{\gamma 1} \mid \right] \quad \left[ \vartheta \parallel \ D_{\gamma 3} \right] \quad \left[ \theta \parallel \ D_{\delta 3} \right] \\ & [\delta \parallel \ D_{\gamma 2} \mid \gamma \parallel \ 1 \oplus \gamma + 1 \Leftarrow (D_{\gamma 2} > 0) \& (D_{\gamma 3} > 0) \right] \\ & \$ \gamma \{\alpha, \vartheta, \theta, \delta \{p\}\} ] \end{array}
```

4. About computation

The accepted structure and forms of representation of information cause not only a compact stratified code, but stratified synthesis and execution, creation stratified semantic check points, more effective location and interpretation of errors, natural interaction, dynamics and so on. Accumalation of knowledge on objects (or a single object) occurs in parallel in various layers and can lead to new and unexpected results (for user of given layer) gotten due to establishment of one-to-one onto functions and mappings (the princple of implicit knowledge).

It is possible to intervent into object structure, in process point any level of representation in any time in case of having sufficient information (*principle of limited all-permission*). It is important to bring out moments and elements of failure into light (to highlight the misunderstood or simply unknown), just here objective and subobjective are interacted.

Process of computing for AS is realized on the base of knowledge of state of environment and knowledge of syntax form. Features of state of compute environment are;

Types of Action	 Local Localised Global	Type of Control	 [Non]Deterministic Sequential Parallel Partial-Ordered
-----------------	---	-----------------	--

All kinds of actions differ by their effect on the computational environment: *local* effects only a previously bounded part; *localised* effects locally that part of environment which have been allocated in a global search process; *global* effects the environment as a rule entirely.

At present the common form of computation is linguistic one, other forms are used as interfaces. Therefore it is being set up the hypothesis about exiting of multidimensional space of representation in which calculations are realized on base of "resonance processes" and more effective than linguistic.

One of the base form of linguistic calculations is goal-oriented synthesisgeneralization of a group of AS with the same structure. Multidimensional representations discriminate synthesis not only by syntax form but by semantic features. Synthesis is mixed with verification and other calculations what allow to react more flexy on local change of compute situation and to control the deep of derivation and area of solution search. In this case parallelism of synthesis in various layers has a natural character.

Here conceptual synthesis is being considered. Expressed on mx language and refined statements of computability, proposed by Tyugu[TE84]:

Statement of Computability

• simple	$A \xrightarrow{f} B$
ullet with condition	$P \Rightarrow A \xrightarrow{g} C$
• with subgoal	$\forall s(DX \xrightarrow{s} EY \Rightarrow RX \xrightarrow{h(s)} GY)$

The first statement correspond with simple axioms, specifinig functions of program system in following form:

 $[A.B (C D)-in E-out] or [A.B (int int_{**})-in graph_*-out]$

If functions realize computation with similar semantic and (or) have the same data parameterized axioms are introduced to decrease the search time and the requested memory for axioms:

$$\llbracket A . \parallel \mathbf{X} \& \mathbf{Y} \Rightarrow \begin{bmatrix} .X & (C \ D) - in & Y - out \end{bmatrix}$$

Here X,Y are free variables in Refal style, "." is concatenation. The process of designing of parameterized axioms is generalization, the process of usage is concretization, being written in the below form:

$$[A \ B \ E], \qquad [A \ N \ S]$$

The base of axioms allow to formulate various goal of search, calculation, costruction of AS:

- $\begin{bmatrix} \pounds & A-in & B-out \end{bmatrix}$ • for function
- $\begin{bmatrix} F & (A \ \pounds) in & B out \end{bmatrix}$ • for data $\begin{bmatrix} \pounds & (C \ \pounds) - in \quad \pounds - out \end{bmatrix}$
- for mixter

Let the sign £designates a goal element, which can be named by the following identifier. The goals are parameterized as is the case with axioms:

$$\blacksquare \pounds . A \parallel \mathbf{Y} \Rightarrow Z \text{-} in \quad Y \text{-} out \blacksquare$$

It is stated here that a wider class of problems than in traditional systems of conceptual programming. Besides, here specifications are executable. Similarily to PRIZ it permits specification of high order functionals in a form of the parameterized axioms with subgoals:

```
\llbracket M \parallel \mathbf{X} \& \mathbf{Y} \Rightarrow \llbracket \pounds \quad X\text{-}in \quad Z\text{-}out \rrbracket \llbracket \pounds \quad X\text{-}in \quad W\text{-}out \rrbracket
```

The derivation rules for compositions and generation of subgoals:

```
Rule.One
    ||-feature mixture
        \| \mathbf{N} \cdot id [C \| 1] \quad [L \| (\mathbf{A} : out \cdot \mathbf{B} : in) \cdot the \neq \emptyset \Rightarrow \{A B\}
            \Leftrightarrow [N ((A+B): in - (A: out \cdot B: in)) - in (A+B): -out]
        \| (\mathbf{A} : out \cdot \mathbf{B} : in) \cdot the = \emptyset \& C = 1 \implies [C \parallel 0] [L \land B]
        \parallel \mathbf{E} - exp \Rightarrow [message \ E]
    \parallel \mathbf{E} \text{-} exp \Rightarrow [message \ E]
□ Rule.Two
    \| ([\mathbf{S} \quad (\mathbf{D} \ \mathbf{X}) - (list \ in) \quad (\mathbf{E} \ \mathbf{Y}) - (list \ out)] \rightarrow \| 
        [\mathbf{H} \quad (\mathbf{R} \ \mathbf{X}) - (list \ in) \quad (\mathbf{Q} \ \mathbf{Y}) - (list \ out) \quad [\pounds]])\&\mathbf{F}
    \Rightarrow [H (R X)-in (Q Y)-out
        \{ [\pounds (R X)-in D-out] [S (D X)-in (E Y)-out] \} \}
    \Leftrightarrow [F \quad (R \ X) - in \quad (Q \ Y) - out]
    \| ([\mathbf{S} (\mathbf{D} \mathbf{X})-(list in) (\mathbf{E} \mathbf{Y})-(list out)] \rightarrow \| 
            [\mathbf{H} \quad (\mathbf{R} \ \mathbf{X}) - (list \ in) \quad (\mathbf{Q} \ \mathbf{Y}) - (list \ out) \quad [\pounds]]) \& D - out \& \mathbf{F}
    \Rightarrow \{ [\pounds (R X) - in D - out] \}
           [H (R X)-in (Q Y)-out [S (D X)-in (E Y)-out]]\}
    \Leftrightarrow [F \quad (R \ X)-in \quad (Q \ D \ Y)-out]
```

Note that the second form of generation of subgoals is somewhat like to planning on undependence tasks of Tyugu, but allows automatic generation of the subgoals. The rules with *if* and *do* are in the traditional form.

References

- [AKP93] Ait-Kaci H., Podelski A. Towards a meaning of LIFE. The Journal of Logic Programming, 16, 1993, pp. 195-234.
- [DGK93] Darlington J., Guo Y., Kohler M. Functional Programming Languages with Logical Variables: A Linear Logic View. Proc. of the Fifth International Symposium PLILP'93 Lect. Notes of Comp. Sci., 714, Springer Verlag, 1993, pp. ???
- [GM87] Goguen J.A., Meseguer J. Models and equality for logical programming. Proc. of TAPSOFT, 250 of Lect. Notes in Comp. Sci., 87, Springer Verlag, 1987, pp.1-22
- [GY93] Guo Y. FALCON: Functional and Logic Language with Constraints. A draft prepared for the 6 of Ph.D thesis: Definition al Constraint Programming. Dep. of Comp. Imperial College, 180 Queen's Gate London SW7 2BZ U.K., 1993.
- [HS86] Hyvonen E., Seppanen J. LISP MAAILMA. 1. Johdatus kieleen ja olyelmointiin, 1986, 2. Ohjelmointimenetelmat ja järjestelmat – Kirjayhtyma Oy, Helsinki, 1987 (in Finnish)
- [HJ93] Haridi S., Janson S., Montelius J., Franzen T., Brand P., Boortz K., Danielsson B., Carlson B., Keisu T., Sahlin D., Sjoland T. Concurent Constraint Programming at SICS with the Andorra Kernel Language (Extended Abstract). Workshop on Principles and Practices of Constraint Programming, ???, 1993.
- [LFB93] Lopez G., Freeman-Benson B., Borning A. Kaleidoscope: A Constraint Imperative Programming Language. In Constraint Programming. NATO ASI Series. Springer Verlag, 1994 (To appear)
- [LM80] Lehman M. Life Cycles, and Laws of Software Evolution, IEEE 68, N 9, September 1980, pp. 1060–1075.
- [MS93] Maslov S. The Program Base is Knowlege Investments. STI, An informational processes and systems. 1, 1993, pp. 1–9 (in Russian)
- [MT92] Matskin M., Tyugu E. The NUT Language. TRITA TCS SE 9212
 TR. Royal Institute of Technology Dep. of Teleinformatics Software Engineering, Stockholm, 1992.
- [PT93] Penjam J., Tyugu E. Constraits in NUT. In Constraint Programming. NATO ASI Series. Springer Verlag, 1994. (To appear)
- [TE84] Tyugu E. Conceptual programming. Moscow, Nauka, 1984 (in Russian)

[TF] Townsend C., Feucht D. Designing and programming personal expert systems. TAB Book Inc. Blue Ridge Summit, PA 17214,