Intent Operationalisation for Source Code Generation

Fumio NEGORO
The Institute of Computer Based Software Methodology and Technology
11-3 Takanawa 3-chome, Minato-ku, Tokyo 108-0074, Japan

ABSTRACT

In the research on software development, there was less achievement in an efficient general development methodology that could be effective and sufficient in dealing with a wide range of software problems related to different domains. Also a challenge of having a universal model for software development was not so easy due to the difficulty of having a general model that can transfer or model a user intention in a well-predicated manner. This is because it is not easy to transfer all the user rational requirement into correct complete specifications. This is due to ambiguity in understanding the requirement, as they exist in a user mental state. This is also due to a mix-up between the requirement and engineering.

In this paper we have succeeded in establishing a universal model that can transfer a user intent expressed in a fashion of natural language. In this way the intent can be represented by words (i.e. nouns) extracted from the expressions in natural language. In order to realize the intent, we have established what we call Predicate Structure (PS), and through it we can materialize a one-variable proposition (i.e. one noun) which takes those nouns as variables. Operation of each variable (i.e. a noun) on the PS can create an instant state of that variable. The representation of each noun in the predicate structure has an autonomous nature. So there is no certain sequence that the PS should follow to realize the user requirement (i.e. intent). That is, by using the PS, the intent can be simply transformed into a set of propositions, which in turn can be symbolized by user-defined programming language. The non-sequential nature of programs for each noun is realized by the PS. We think this can eliminate the ambiguity in the realization of the intent behind the user requirement. This is proved with our theoretically derived Three-Dimension-like Space Model. To establish complete requirement specifications, we have built an algorithmic procedure called Scenario Function. This, in its recursive nature, can re-call itself till the user meaningful intention is realized. What we get on a computer screen is regarded as his/her intention, which exists before its formation.

This shows a new way of looking at software development and its realization can simplify the procedure of software development. With this methodology, we could also reach less development time in comparison with normal software development procedures. With such a structural approach of software, we could transform any source program into a predicate structure. In this way, the conventional structured program can be restructured. This leads to a new style of re-engineering in a much simpler and efficient manner, in comparison with re-engineering in an object-oriented framework.

Keywords: Predicate structure, tense control vector, control structure, hypothesis, synchronous state, scenario function, development methodology

1. INTRODUCTION

Lyee[1] is a generic name given to the methodology of modeling non-objectified intent axiomatically as well as the methodology of modeling objectified intent by the axiom. A model based on the former methodology is called Consciousness Model, and a model based on the latter methodology is called Cognitive Model. With this subject, an observation is made on Cognitive Model (called this model hereinafter) which is symbolized by programming language.

1.1 Harmonization

An intent can be replaced by a set of propositions that can be defined by Predicate Structure which materializes based on the Lyee axiom system (called simply this axiomatics, hereinafter). And, this Predicate Structure becomes definite against programming language which symbolizes an intent and its propositions as well. This means that this Predicate Structure can harmonize with optional intents.

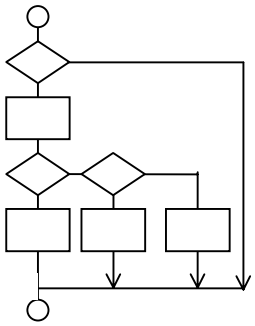


Fig. 1: Predicate Structure

A structure which materializes an intent is largely classified into the following two structures. That is,

- Predicate Structure to materialize one-variable proposition taking a noun as a variable
- Control Structure to materialize the sequence of the proposition on a computer

Predicate Structure is shown in Fig. 1. Control Structure is omitted.

1.2 Complementary Action

Predicate Structure materializes, based on a hypothesis, as an instant state in which an intent materializes. It is a state in which verbs do not materialize. Therefore, this Predicate Structure which is introduced from this basis can realize significant elements so as to regulate an entity of existence only by nouns contained in natural language. The role of verbs is absorbed into the Predicate Structure materializing a proposition and the complementary action between propositions.

The complementary action means that propositions executed based on this Control Structure autonomously symbolizes the mutual existence relationship between propositions as data, respectively. In this connection, an action to sequence process is an important role of verbs. The complementary action substitutes it. Because of this, with the Lyee program, the problem of the process sequence is eliminated.

On the other hand, with traditional programs, it is necessary to settle in advance the process sequence problem by using knowledge and experience concerned with an intent. Nevertheless, it becomes self-willed. The process sequence is, an imperative part of logic. Therein, the process sequence becomes indispensable, but with Lyee it is unnecessary.

The data combination of traditional programs materializes asynchronously because it materializes based on the process sequence. On the other hand, the data combination of Lyee materializes not based on the process sequence, so it materializes synchronously.

[1] Lyee stands for GovernmentaL methodology for software providence, invented by Fumio Negoro.

1.3 Tense Control Vector

The proposition of this model is classified into nine kinds. The three kinds are a one-variable proposition, the rest six kinds are not a one-variable proposition but can be defined by the same structure as the Predicate Structure of one-variable proposition. The three kinds of one-variable proposition realize data combination complimentarily and synchronously. The one-variable proposition is generally called Signification Vector. The state in which a result is existent in the 4th box of Fig. 1 is when the proposition is TRUE. The proposition in that state realizes complementary and synchronous data combination.

On the other hand, current computers have a scheme to realize asynchronous data combination in accordance with a way of our cognition. For this reason, the six kinds of propositions are prepared for performing roles to materialize the harmonization of the synchronous data combination realized by the three kinds of one-variable proposition and its asynchronous data combination realized by current computers.

Tense Control Vector		Pallet		
		W04	*W02*	*W03*
Signification Vector	For a defined input word		(5)	
	For a defined output word	(1)*		(8)
Action Vector	Input Vector		(6)	
	Output Vector	(2)		
	Structural Vector	(3)		
	Route Vector	(4)	(7)	(9)

* (1), for example, is the signification vector for a defined output word on a *W04* pallet.

Table 1. Placement of Tense Control Vectors

The six kinds of proposition are generally called Action Vector. The nine kinds of proposition are generally called 'Tense Control Vector'.

All the Tense Control Vectors are not only universal with their structure but also universal with their contents. Resultantly, the definition can be implemented independently with one another. Tense Control Vector becomes the element of three kinds of a set. These sets are expressed by *W04*, *W02* and *W03* and called Pallet.

The concept of Pallet is defined based on the Lyee axiomatics. The relationship of the placement of Pallets and the nine kinds of Tense Control Vectors are shown in Table 1.

1.4 Control Structure

The complementary relation which materializes between Tense Control Vectors makes other Tense Control Vectors TRUE one after another upon the momentum of Tense Control Vector which becomes TRUE. Therefore, Tense Control Vectors with a relationship of materializing the complementary relation are gathered into a set, and the set is executed in repetition. By this, Tense Control Vectors in the FALSE state moves to the TRUE state.

The gathering of Tense Control Vectors into three kinds of Pallet is ruled by the Lyee axiomatics. Programs which constitute the Control Structure are classified into two kinds. That is, the control program which executes Pallets in repetition and the control program which executes Tense Control Vectors belonging to Pallets in repetition. The former is called 'Tense Control Function' and the latter 'Pallet Function'. These two kinds of programs are defined with no relation to an intent. The relation between Control Structure and Tense Control Vector is called 'Scenario Function'.

1.5 Hypothesis

The Lyee axiomatics materializes in the Consciousness Space. The Scenario Function materializes in the Three-Dimension-Like Space Model which is delivered by the axiomatics. Herein, the main point of this Three-Dimension-Like Space Model is explained.

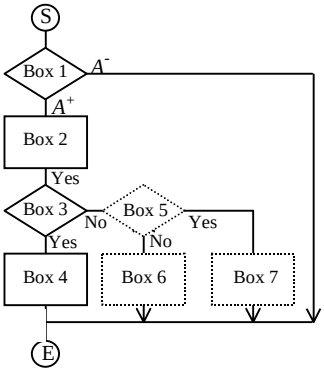
The three kinds of Pallet comprise three kinds of coordinates, and materialize the Three-Dimension-Like Space. In this space, two spaces (A^+, A^-) with different nature co-exist. One space is expressed by (A^+) and called Consciousness Space, and the other space is expressed by (A^-) and called Natural Space. The three kinds of Pallet are complementary spaces which combine Consciousness Space and Natural Space. In Consciousness Model, five kinds of space are defined.

Predicate Structure (Fig. 1) which materializes Signification Vector is what has replaced the algorithm, which materializes Consciousness Space and Natural Space, by the Three-Dimension-Like Space Model. The algorithm of its cause materializes in Consciousness Model. It is shown in Fig. 2. The Three-Dimension-Like Space Model is a typical presentation of three kinds of Pallet, Signification Vector, Consciousness Space and Natural Space. It is shown in Fig. 3.

If Signification Vectors with the same variable and belong to more than two kinds of Pallet become TRUE altogether, either element of Consciousness Space or Natural Space is indicated by those Signification Vectors. The number of elements of both spaces is infinite respectively, but the number is always greater in Consciousness Space. When the number of elements of Cognitive Space and that of Consciousness Space are determined, a corresponding relationship is established between these spaces.

When the corresponding relation of both spaces materializes, elements belonging to Natural Space are objectified. Those objectified elements become recognizable to us, whereas the yet-to-be objectified elements in the two spaces are intrinsically out of our recognition.

In the 1st box of Fig. 2, it is inquired which space's element the indicated element is. If it is an element of Consciousness Space, the 2nd box objectifies its corresponding element of Natural Space.



Box	Instructions in the box
1	Which does a subset of the elements (Logical Atoms) belong to, A^+ or A^-?
2	Objectify one of the elements of the subset in A^- corresponding to the subset in A^+ so as to make that element a substance of the existence.
3	Has the objectification been done?
4	Objectify the rest of the elements of the subset in A^- so as to make them attributes of the existence.
5	(Explanations on dotted boxes are omitted.)
6	
7	

Fig. 2: Structure of the Principle of Objectification

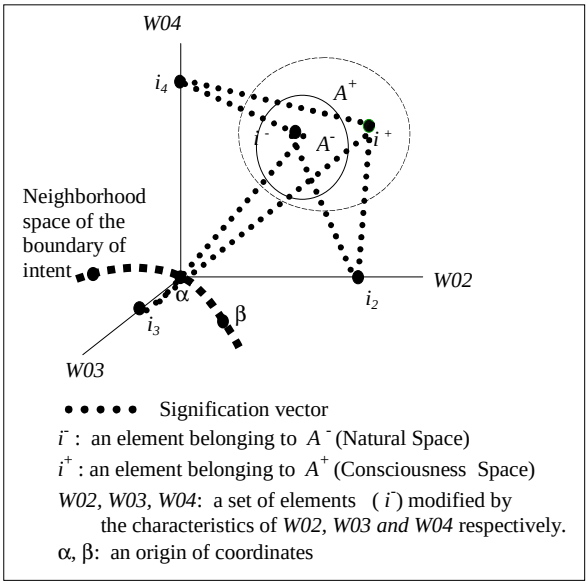


Fig. 3: Three-dimension-like Space Model

As the space has already been replaced with Natural Space, it is inquired in the 1st box of Fig. 1 if the element has been objectified or not. If not objectified, the objectification corresponding to the element, such as arithmetic operation, is implemented in the 2nd box. If objectified, it means that the cognition of an intent has already been established. Therefore, there is nothing more to be done.

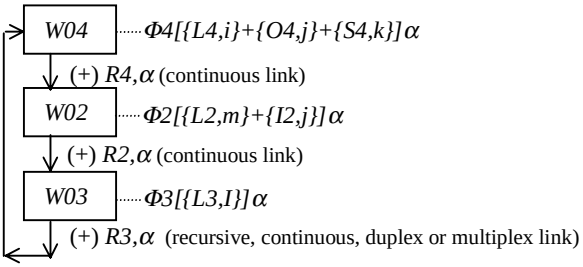
In the 4th box of Fig. 2, the element of Natural Space is made into the ATTRIBUTE. The 4th box of Fig. 1 makes the action of the 2nd box recognizable.

1.6 Scenario Function

Our cognition of an intent is realized by the materialization of a state which synchronizes with an intent by objectification. Scenario Function assumes this role. A computer assumes roles of the implementation of this Scenario Function, the establishment of a synchronous state in the memory area, and the establishment of an asynchronous-like synchronous state that materializes the cognition of an intent with us.

Scenario Function for a whole set: If the whole set of an intent is designated by U, the Scenario Function is expressed by $T_{(U)}{}^{1}$, and defined as follows:

$$T_{(U)}{}^{1}=\Phi\,(\sum_{1}^{\alpha}[W04+W02+W03]\alpha)$$



Since an intent can be grasped by plural pieces of Scenario Function, each Basic Structure is specified with an identifier, α. Symbols are explained below.

i. An intent is expressed by the link state of plural Scenario Functions. This is made into diagram and called Process Route Diagram. Symbol to couple Scenario Functions: *Σ*. The link materializes with four kinds of rules (Link Rules). Details are explained later.
ii. Symbol to express the execution sequence of Pallet and Tense Control Vector: +
iii. A set of taking Tense Control Vector as element: *{}*
iv. Tense Control Function: *Φ*
v. Pallet Functions belonging to each Pallet: *Φ4, Φ2, Φ3*

Tense Control Function and Pallet Function can be defined with no relation to an intent, once programming language of the execution environment such as OS is determined.

vi. Signification Vector of the input attribute word belonging to *W02*: *L2*
vii. Signification Vector of the output attribute word belonging to *W03*: *L3*
viii. Signification Vector of the output attribute word belonging to *W04*: *L4*

Data items of traditional programs are called Word with Lyee.

ix. Output Vector: *O4*
x. Input Vector: *I2*
xi. Structural Vector belonging to each Pallet: *S4*
xii. Route Vector belonging to each Pallet: *R2, R3, R4*

Scenario Function for a part: If a part of the whole set of the intent *U* is designated by *A*, the Scenario Function of an intent *A* materializes based on its instant intent A_i and expressed by $T_{(Ai)}{}^{1}$.

The A_i is a subset of *A*. The $T_{(U)}{}^{1}$ is definable but impossible to materialize it. In contrast, the $T_{(Ai)}{}^{1}$ is possible to define and materialize. The $T_{(U)}{}^{1}$ and $T_{(Ai)}{}^{1}$ have the following relation. That is, $[T_{(U)}{}^{1}\rightarrow A]=T_{(Ai)}{}^{1}$, and this relation is called the Harmonization Formula.

The relation of traditional programs and Scenario Function: A result of execution of Scenario Function on a computer is a result of repeated execution of $T_{(Ai)}{}^{1}$. Therefore, the result is expressed by $M(T_{(Ai)}{}^{n})$. On the other hand, the execution result of traditional programs is $M(P_{(Ai)})$. Although an intent to be realized by a traditional program is a subset denoted as *(Ai)j* which consists of subsets, *Ai*'s, of *A*, a traditional program is simply denoted as $P_{(Ai)}$.

As for $P_{(Ai)}$ and $T_{(Ai)}{}^{n}$, if they are what satisfy the same intent respectively, the execution result becomes equivalent in view of interpreting a meaning. This is expressed by $M(P_{(Ai)})\equiv M(T_{(Ai)}{}^{n})$ and called Synchronization Formula. It is important that the program structures of $P_{(Ai)}$ and $T_{(Ai)}{}^{1}$ are totally different. That is, the Scenario Function is in a static state whereas a traditional program is in a dynamic state.

Software expressed by Scenario Function: Scenario Function is the software that symbolizes an ambiguous intent. In this connection, the arithmetic expression defined in advance and its algorithm do not contain ambiguity in themselves, even if they are expressed in programming languages, so they are intrinsically fixed and are not software discussed herein.

If the arithmetic expression is delivered during execution process and the algorithm calculating it is delivered during execution process, the arithmetic expression and its algorithm are software discussed herein.

Till now, the software definition has been ambiguous from macro viewpoint. As a result, standardized functional components and even calculation algorithm have been added to the category, but in order to genuinely advance software, they must be separated originally.

1.7 Development Methodology

The development methodology discussed in this subject is a work algorithm to determine Scenario Function.

Work items: . Items of work are as follows:

i. Definition of user's Definitive
ii. Definition of words
iii. Determination of Process Route Diagram
iv. Definition of logical units
v. Definition of variables

vi. Definition of Vectors
  vi-1) Output Vector
  vi-2) Input Vector
  vi-3) Structural Vector
  vi-4) Route Vector
vii. Definition of Signification Vector
  vii-1) *W04* Signification Vector
  vii-2) *W02* Signification Vector
  vii-3) *W03* Signification Vector
viii. Systemization of Scenario Function onto a computer
  viii-1) Definition of physical units
ix. Compressed Simplification of Process Route Diagram

To replace an intent with words in view of user's standpoint means, for example, Screen (display) and Voucher are defined. With Lyee, this is called User Definitive (or simply, Definitive). The Definitive becomes a logical unit when it is redefined for software and it becomes a physical unit when it is redefined for a computer system.

The number of Scenario Function corresponds to the number of logical unit. Normally, an intent is expressed by plural logical units. Therefore, an intent is expressed by the state of the coupled plural Scenario Functions. That is, Process Route Diagram. The Process Route Diagram is a space structure which makes an intent axiomatically.

An example of Process Route Diagram is illustrated in Fig. 4. The Process Route Diagram is a document to determine Scenario Function. Based on the determination of Process Route Diagram and the information of defined logical units, a word to be defined is positioned as a variable of *W02* Signification Vector, if its attribute possesses input attribute, or as a variable of *W03* and *W04* Signification Vector, if it possesses output attribute. The input logical unit is placed in *W02* and the output logical unit in *W04*. Basic Structures with the same nature can be put together. This is called Compressed Simplification.

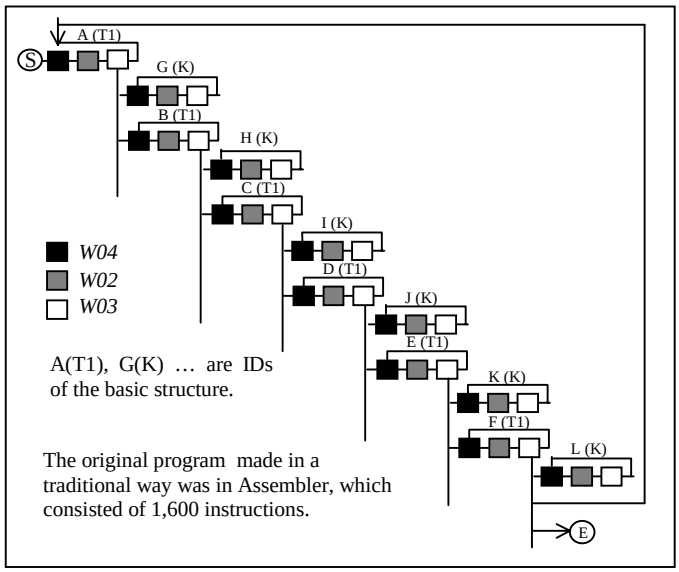


Fig. 4: Process Route Diagram (example)

If Process Route Diagram is determined, algorithm to define Action Vector becomes universal on its base. Algorithm to define Signification Vector is universal on the basis of user's definition information. By using Scenario Function, verification of logic, which is done for conventionally-made programs, is no longer necessary.

Efficiency: Software development work an entire range from the traditional upper stream to lower-stream can be replaced with work algorithm which defines Scenario Function. With this, universality materializes, and for this reason the work efficiency improves. In view of this, the work to define Scenario Function can be rephrased by Software Development Methodology.

2. IDEA OF LYEE

2.1 Circumstances of traditional methods

Circumstances of traditional methods are summarized as follows:

i. An intent of software is symbolized by natural language. Therefore, the interpretation of its meaning becomes personal based on experience and knowledge woven around an intent. In that sense, an intent becomes ambiguous inevitably. It is a big difference compared with physical structures that can make recognition public by making details of constituent elements into a unit.
ii. It is impossible to establish universal epistemology or universal development methodology of software by formalizing the meaning expressed in natural languages with logic or by means of a thinking method of recognizing physical structure.
iii. Therefore, program specifications to determine a model for an intent and its programming method are implemented with personal self-will. As a result, the determination of the program specification and the program becomes self-willed.
iv. The inefficiency of software development work is attributed to the fact that a universal work method cannot be established therein.

For example, to verify the rationality of an intent, the specification must be verified. To verify the specification, the program specification must be verified. To verify the program specification, programs must be verified. In principle, it is difficult to satisfy these verifications on the basis of traditional methods and their extensions.

2.2 Self-will

To symbolize an intent in natural language means to obtain subsets from among sets of countlessly existing words and give sequence to the words belonging to the subsets. This represents a subject of corresponding infinite number with finite number. This subject is out of scope of this paper and will be discussed in another paper focused on the theory.

Therefore, this action is originally implemented in the situation of multiplicity of meaning. We can only manifest it as the one and only. This makes a way of symbolization personal, and it becomes self-willed as a result.

The Lyee study aimed to obtain a method of grasping the inevitable involvement of self-will that is existent in an intent by determinism. The self-will is an entity taking deep root in a way of our recognition. Verbs are deeply involved therein. Therefore, Lyee overcomes it by obtaining a Predicate Structure which excludes verbs from the natural language that symbolizes an intent.

2.3 Universality

The universality cited by this theory means that anyone can symbolize an intent in the same way as personal intent is duplicated. For example, if an intent is A_i, another intent B_i which symbolizes A_i exists and the B_i is established public under certain circumstances.

With Lyee, the intent B_i is an axiom of Lyee (theorem). Scenario Function is established by this axiom (theorem). The materialization of universality in Scenario Function is based on this reason.

2.4 Impression of Lyee program

In the following two sentences nouns are 'I' and 'human'.
I am a human.
I see a human.

With Lyee, these nouns are distinguished into four pieces. By using them as a variable respectively, Signification Vector is defined as regulated, and then Scenario Function is defined. When the Scenario Function, compiled by operator, is executed on computer, an execution result of the Scenario Function is symbolized into the memory area. The operational will of the operator and a state of the memory area let the operator, for example, be aware of two different verbs (AM, SEE). Because of this, the operator becomes aware of the difference of the two sentences.

With Lyee, an intent is treated to be uncertain permanently. Therefore, no verification of the correctness of the specification which grasps an intent is implemented. However, the purpose of software uttered by an intent can be obtained by means of Scenario Function.

3. DETAILS OF CONCEPT

3.1 The relation between traditional program and Process Route Diagram

The input command concerned with the display Screen, for example, of traditional program and the output command concerned with File correspond with Scenario Function as 1 versus 1. Therefore, commands like them are called asynchronous commands, with Lyee.

The execution sequence of asynchronous commands in traditional program is used as a state of the link of plural Scenario Functions, that is, as information to define Process Route Diagram. An individual Scenario Function in the Process Route Diagram is particularly called Basic Structure.

3.2 Link of Basic Structures

The relation of *W04* and *W02* in the same Basic Structure is an asynchronous relationship. Therefore, Signification Vectors of *W04* and *W02* cannot commonly share the state of respective memory areas. However, relations of *W04* and *W03* as well as *W02* and *W03* are a synchronous relationship, respectively, so the state of respective memory areas can be commonly shared. A route from *W04* to *W02* in the same Basic Structure is called Continuous Link, and a route returning from *W03* to *W04* is called Recurrent Link.

Continuous Link and Recurrent Link are an action to establish a synchronous state in a Basic Structure. In contrast, in neighboring two Basic Structures, a link returning from the latter-positioned Basic Structure to *W03* of the former-positioned Basic Structure is called Duplex Link.

This link establishes an expanded synchronous state in the former-positioned Basic Structure. The two Basic Structures with a relation of Duplex Link establish a synchronous relationship between them. Basic Structures connected with Continuous Link are asynchronous. With the link relation, there are the following cases: to establish additional Duplex Link from the Basic Structures with a Duplex Link relation, or to couple two Basic Structures by Continuous Link.

Also, there is a case in which the latter-positioned Basic Structure couples with the former-positioned Basic Structure by another link method but not by Duplex Link. In case of Duplex Link, it is a route to return to the former-positioned *W03*, however, in this case it becomes a route to return to the former *W04*. This link is called Multiplex Link.

The next Pallet is determined by Route Vector which belongs to respective Pallets, and, based on the information, the Tense Control Function hands over the execution control to the Pallet Function of the next Pallet. The conditions to determine the next Pallet which are used by respective Route Vectors are shown in Fig. 5, the principle of links.

3.3 Words

With Lyee, words belonging to Definitive are called Regular Word. Based on their input or output attribute, the Regular Word is placed in each Pallet of the Basic Structure in accordance with the aforementioned rule. According to this, necessary memory areas are determined. Then, the word becomes a variable and materializes Signification Vector. The determined memory areas are used for the actions of Signification Vector. The logical unit to which the word becoming a variable belongs is called the 1st coordinates of word.

With traditional program, there is an item called *K* Word in addition to Regular Words. With traditional program, internal logic is indispensable and this word is prepared for that purpose.

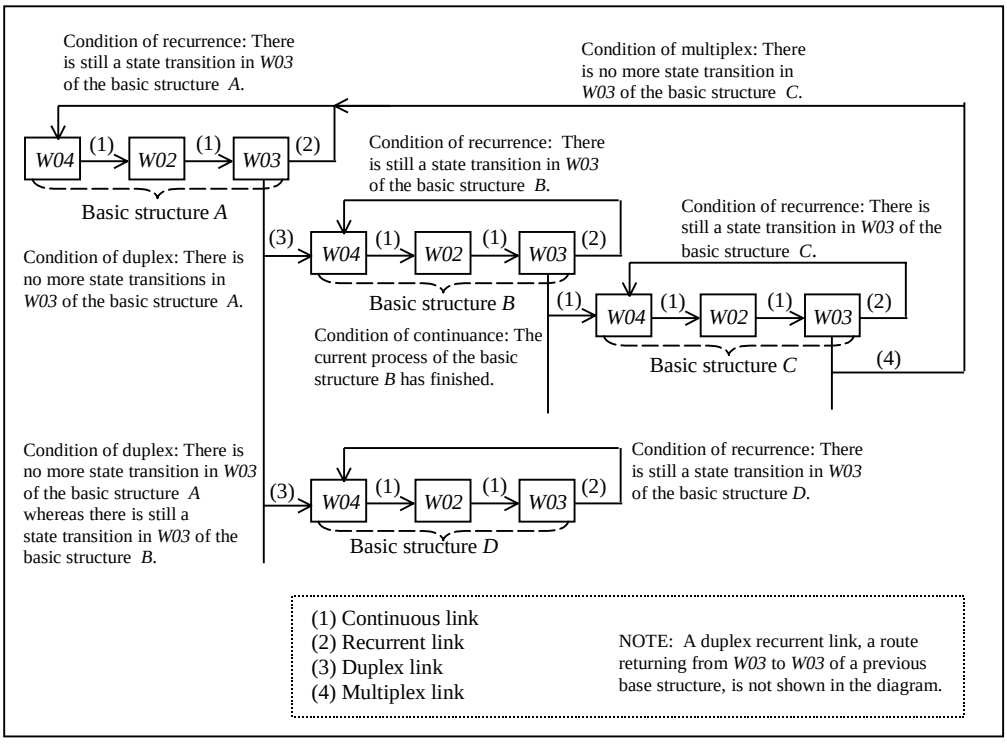


Fig. 5: Principle of Links

A variable belonging to *W04* of the Basic Structure happens to become a variable of *W04* of the same Basic Structure after being duplicated. A word of such nature is called Equivalent Word. In this instance, Signification Vectors of *W04* shall be defined in the same number of the materialization of Equivalent Words. However, their variable names are distinguished. The *W03* Signification Vector in that instance is one piece. When the Signification Vector of this *W03* becomes TRUE, it means that the designation is made to the Signification Vector that shall be executed from among plural Signification Vectors of *W04*.

A variable that belongs to

Pallet	Type of vectors (1)	Tense control vectors (1)	Self word area				Supple. area		Control box area					
			Box 2	Box 4	Box 6	Box 7	State transition table	Suppl. area for a self word (2)	Logical unit	Status	Current access key	Previous access key	EOF	MSG FLG
W04	SV	1. SV for output word & state control vector	● (3)	●	●	●	●	◊ (4)	(5)					
	AV	2. Output vector & duplication vector	●		●	●			●	●	◊	◊	●	●
		3. Structural vector	●	●										
		4. Route vector		●										
W02	SV	5. SV for input word & state control vector	●	●	●	●	●	◊						
	AV	6. Input vector	●		●	●			●	●	◊	◊	●	●
		7. Route vector		●										
W03	SV	8. SV for output word & state control vector	●	●	●	●	●	◊						
	AV	9. Route vector		●										

(1) SV and AV denote the signification vector and the action vector respectively.
(2) This area is required for a word to be used for summations.
(3) A dot denotes that the memory area is required.
(4) A diamond denotes that the memory area is required in certain cases.
(5) A shaded box indicates that the memory area is not necessary.

Table 2: Tense Control Vectors and Memory Areas

another Basic Structure after being duplicated is called Boundary Word versus the original word. Differences from normal *W04* Signification Vector are that, when the Signification Vector of the original word becomes TRUE, the Signification Vector of its Boundary Word is also made TRUE.

3.4 The memory area of Tense Control Vector

The memory area required by Signification Vector and Action Vector is explained herein. Refer to Table 2 (Tense Control Vectors and Memory Areas). In order to materialize Tense Control Vector, memory areas are prepared in correspondence to commands placed in its 2nd box, 4th box, 6th box and 7th box. Prepared in addition to these areas are the input/output area (logical unit) for Input/Output Vectors and the memory area (called status area) which captures the after-execution state of input/output commands. They are called Control Box Area (CB Area).

In case the 2nd box command takes a format of accumulation, a supplementary area, which is required for a word to be used for summations, is added exclusively for the summation. In the 6th box, a command which sets a directive information (Refusal Flag) is placed to stop its Restart. In the 7th box, a command which sets a directive information (Restart Flag) is placed to implement Restart. Therefore, memory areas are prepared which maintain directive information corresponding to the 6th and 7th boxes. In the 5th box, if there are changes in the state compared before and after all Refusal Flags of the Pallet, the judgment of Restart is implemented, and if no changes in the state, the judgment of Restart Stop is implemented. In the 1st box, in order to make judgment of execute or not, a command which makes judgment on Restart Flag is placed.

3.5 Signification Vector

Explanation on Signification Vector is made herein.

***W02* Signification Vector:** The Signification Vector placed in *W02* is defined by taking input attribute word as a variable. And, it assumes a role for setting input logical unit's information into the 4th box memory area of this Signification Vector.

The 1st box command inquires if information exists in the 4th box memory area. If no information in the 4th box memory area, information is obtained from input logical unit by the 2nd box command, and it is once made to exist in the 2nd box memory area. The 3rd box command verifies an attribute of the information existing in the 2nd box memory area. If the attribute matches, the information of the 2nd box memory area is set into the 4th box memory area. The time this Signification Vector becomes TRUE is in this state.

If the result of the attribute verification is not proper, controls of this Signification Vector move to the 5th box command. The 5th box command makes judgment of executing this Signification Vector again or not. However, if the result of the attribute verification is not proper, the cause is not attributed to this Signification Vector, but becomes a problem of input logical unit information itself.

Therefore, this problem shall not be solved even if this logical unit is executed in repetition, unless the input logical unit is changed. Thereupon, the Refusal Flag is set in the 6th box memory area by the 6th box command. The Refusal Flag and the Restart Flag which is set in the 7th box memory area by the 7th box command take a complementary relation, so, the setting of the Refusal Flag is the same thing as the resetting of the Restart Flag.

In the 1st box, the state of the 4th box memory area and the state of the setting of Restart Flag (the resetting of the Refusal Flag) are inquired by the OR relationship. Therefore, if information already exists in the 4th box memory area, or if the Restart Flag has been reset, this Signification Vector ends without being executed.

When an attribute matches and information of the 2nd box memory area is set into the 4th box memory area, the Restart Flag is reset by the command in the 4th box. The Restart Flag has been set by the initial state (value).

***W03* Signification Vector:** The Signification Vector placed in *W03* is defined as a variable by taking output attribute word. It verifies go-no-go of the execution of *W04* Signification Vector with the same variable, and assumes a role of setting the judgment result into the 4th box memory area. The *W04* Signification Vector checks on the judgment result which has been set, by the 1st box command, into the 4th box of *W03* with the same variable, and autonomously acknowledges go-no-go of the execution.

The 1st box command of the *W03* Signification Vector inquires if the judgment result already exists in the 4th box memory area. If the judgment result does not exist, a judgment information is defined by the 2nd box command. It is judged by the 3rd box command. If executable, the judgment result is set into the 4th box memory area by the 4th box command. When this Signification Vector becomes

TRUE is at the time of this state.

If the 3rd box judgment is not executable, the control of this Signification Vector moves to the 5th box command. The 5th box command makes judgment if this Signification Vector must be re-executed or not.

The 5th box command judges on any presence of the 4th box's before-and-after status change of all *W03* Tense Control Vectors. If there are any changes, the Restart Flag is set. If no changes, the Refusal Flag is set. The relationship of Restart Flag and Refusal Flag is the same as the case of *W02*.

Also, same as the *W02* case, in the 1st box, in addition to the 4th box memory area's status, the OR relation is inquired to the setting of Restart Flag (the resetting of Refusal Flag). Therefore, if information already exists in the 4th box memory area, or if the Restart Flag has been already set, this Signification Vector ends process without being executed. In case the judgment result is set in the 4th box memory area, the Restart Flag is reset by the 4th box command. The Restart Flag has been set by the initial value, same as the case of *W02*.

***W04* Signification Vector:** The Signification Vector placed in *W04* is defined as a variable by taking output attribute word. It assumes a role of setting information of the output logical unit into the 4th box memory area of this Signification Vector.

The *W04* Signification Vector checks on the judgment result which has been set, by the 1st box command, into the 4th box memory area of W03 with the same variable and autonomously acknowledges go-no-go of the execution. If executable, the variable information is created by the 2nd box command, and it is once put into the 2nd box memory area for existence. The 3rd box command inquires the presence of information in the 2nd box memory area. If information exists, the information of the 2nd box memory area is set into the 4th box memory area. When this Signification Vector becomes TRUE is at the time of this state. If information does not exist, the control moves to the 5th box command. The 5th box command judges if this Signification Vector must be re-executed or not. The 5th box command judges on any presence of the 4th box's before-and-after status change of all *W03* Tense Control Vectors. If there are any changes, the Restart Flag is set. If no changes, the Refusal Flag is set. The relationship of Restart Flag and Refusal Flag is the same as the cases of *W02* and *W03*.

Also, same as the cases of *W02* and *W03*, in the 1st box, in addition to the 4th box memory area's status, the OR relation is inquired to the setting of Restart Flag (the resetting of Refusal Flag). Therefore, if information already exists in the 4th box memory area, or if the Restart Flag has been already set, this Signification Vector ends process without being executed. In case the judgment result is set in the 4th box memory area, the Restart Flag is reset by the 4th box command, same as the cases of *W02* and *W03*. The Restart Flag has been set by the initial value, same as the cases of *W02* and *W03*.

3.6 Action Vector

Action Vectors are explained hereunder.

Input Vector: This vector implements an action to duplicate physical unit into logical unit. This Vector is defined for every logical unit and placed in *W02* of the Basic Structure. When the same Input Vector is defined for plural Basic Structures, if one of those Input Vectors becomes TRUE, the same Input Vectors are also made TRUE by the first-TRUE-state Input Vector.

Output Vector: This vector moves the state of the Self Word memory area to logical unit and implements an action to output on external devices. This Vector is defined for every output logical unit and placed in W04 of the Basic Structure.

Structural Vector: This vector implements after-process against the Destruction of Synchronization caused by the Output Vector, that is, Memory Area Continuation, or initialization process. The Structural Vector is created for every classification of the memory area prepared in Pallet and placed in *W04*.

Route Vector: The Pallet change control is implemented by Tense Control Function. The Route Vector creates information which is required for that purpose. Even if plural change-to Pallets exist, the Route Vector is prepared collected into the one-piece relation per Pallet.

3.7 A set of Tense Control Vector and process in repetition

Assuming three kinds of words, *a*, *b*, and *c* are in the relationship of $a=b+c$, from this relation, the word *a* is grasped as a word possessing output attribute. Therefore, the second coordinates of word *a* becomes *W04*, and becomes a variable of *W04* Signification Vector. This relation is defined by its 2nd box command. That is, based on the rule of the Predicate Structure area, if the supplementary memory area of the word *a* is defined as *a0*, it can be defined as command $a0=b+c$. Zero behind *a* is an identifier of the supplementary memory area. In this relation, word *a* is called Self Word, and words *b*, *c* are called Given Word.

In this instance, if Given Word is a word with input attribute, the second coordinates of the word becomes *W02*, and becomes a variable of *W02* Signification Vector. That is, the value of the Given Word which realizes Command $a0=b+c$ will exist in the 4th box memory area of *W02* Signification Vector taking Given Word as a variable.

In this instance, if Given Word is a word with output attribute, the second coordinates of the word becomes *W04*, and becomes a variable of *W04* Signification Vector. That is, the value of the Given Word which realizes Command $a0=b+c$ will exist in the 4th box memory area of *W04* Signification Vector taking Given Word as a variable.

If the second coordinates of the Given Word that materializes the 2nd box command of *W04* Signification Vector, the synchronous relation is established between words, so the value of the memory area can directly utilized. However, if the second coordinates of the Given Word is a variable of *W02*, the synchronous relation is not established between the words due to the Pallet relationship, so, it is placed in *W04* as Boundary Word. And, when the *W02* Signification Vector sets a value of the 2nd box memory area into the 4th box memory area, it also sets a value into the Boundary Word area, concurrently. By making Boundary Word as Given Word, the asynchronous problem is solved.

In addition to the output word placed in *W03*, the input word is also placed, and Signification Vector taking it as a variable is defined, which may as well be used as a replacement of the Boundary Word. In this instance, the word placed in this *W03* becomes a Given Word. Likewise, the asynchronous problem is solved.

When Scenario Function is executed, a set of Signification Vectors of each Pallet is executed in repetition by the action of the Pallet Function until leading to no change in the set in the 4th box memory area. When it becomes with no change, it means that its role during the execution cycle of each Pallet has ended. Then, the control moves to the next Pallet. With Pallets, controls move as follows: *W04*→*W02*→*W03*, and with Return, to *W04* of the same Basic Structure, and with Continuous, Duplex, and Multiple, to *W04* of the different Basic Structure. This execution is repeated. By this, a Complementary Action is established between Tense Control

Vectors in a broad sense and between Signification Vectors in a narrow sense.

The relation in which the Complementary Action is established by the repeated action means to guarantee the relation in which the rest Tense Control Vectors also become TRUE. In other words, consequently, the following relation is established in the world of Pallet: In Pallet, for example, if proposition *A*, then in *B*, if *A* is TRUE, *B* as well always become TRUE. This guarantees that the materialization of the part always materializes the whole.

In the world of software, on the premise of the inevitable materialization of the whole, a relation for the establishment of the part has been sought for, however, the structure of Lyee sets ourselves free from it.

4. CONCLUSION

The problem concerned with software development originates in the lack of the recognition of software itself. The start of the problem falls onto the fact that the definition of software itself is insufficient. It is thought that a new conception is indispensable which can be called genuine software science that surpasses traditional engineering viewpoints. For example, matters listed hereunder are concepts different from traditional ones. It ultimately means that the definition of software has newly been advocated.

(1) With Lyee, the development intent is grasped by the set of words. The word can be obtained as a common information that has been made into a unit and materializes between humans and computer, control devices and computer, and humans and control devices.
(2) The materialization of the Signification Vector changes traditional chaotic program structure into an orderly structure.

Because of this effect, traditional designing and manufacturing steps can be transformed into dramatic improvements. For example, works of creation and verification of traditionally so-called internal logic can be reduced sharply. The issue brings out changes in our thinking method in the software development work.

In this connection, effects realized by the second-round use of Lyee will find the greater ratio of the improvement, compared with the first-round use. The productivity of software development work including designing work as well as the efficiency of maintainability improves to the extent far beyond, as incomparable with traditional methods. The situation is more than sufficient to dislodge our dilemma.

Also, the significance that the algorithm can be obtained which automatically converts traditional programs into programs with static structure means the following. In the world of software, it is good news that, we, who can only establish the cognitive method of grasping a dynamic state, are now able to establish a cognitive method of grasping a static state. Because of it, we are now in a position to place ourselves side by side with the common knowledge of the productivity of physical worlds.

This paper will appear in the proceedings of SCI 2001 and ISAS 2001 to be held in July 2001 in Orlando, FL, USA.