A Proposal for Requirement Engineering

Fumio Negoro

The Institute of Computer Based Software Methodology and Technology
3-11-3 Takanawa, Minato-ku, Tokyo 108-0074, Japan
f-negoro@lyee.co.jp
http://www.lyee.co.jp

Abstract. In what psychological state are requirements for a software model determined? It is thought that the state must be existent in the incomprehensible depths of our consciousness. The state is only one as a whole. This subject defines it as an intent. The action that produces a requirement or the action that determines it is dispatched from an intent. In that sense, a requirement reflects an intent. The reflection means the objectification. For example, the expression in language means it. However, its scheme which is implemented as a phenomenon of life shall become none of which is the same. On the basis of the aforementioned assumption, this subject pursues a standpoint of grasping an intent but not of grasping a requirement. For this sake, hypotheses are accumulated. From the said hypothesis, the rule of theorem is sought, and, by using it, an intent is sought. Scenario Function is what was given its concrete form. The work to define the Scenario Function replaces so-called traditional software development itself. The definition can be implemented with the work procedure of determinism. This represents qualifications of the methodology of software development. Related achievements are already available, too. In observing the state of the work, it is undoubtedly effective, when compared with the traditional standpoint. It is thought that it is because of the rationality of the hypothesis of this subject. The purpose of this subject is to describe the overview of the Scenario Function and its hypothesis and publicise that such thinking can be established.

1 Introduction

A universal structure exists in which a set produced on the basis of some mind and nouns belonging thereto can be defined by programming language, and it is called Scenario Function, or SF. When SF is executed on the computer, the action of those nouns autonomously complements with one another and establishes systematisation of significant data among nouns into the memory area of the computer. The state reflects some mind different from a requirement.

In this subject, this mind is defined as an intent. If an intent is made into more abstraction, it comes to an existence. Therefore, it is speculated that an intent dwells within thereabouts, not relating to a state of existence. That is, an intent dwells in the concerned party who knows the systematisation of data in the memory area. The concerned party behaves by reflecting it. In this subject, this composition is defined as software.

In order to define the concept of an intent axiomatically, we build a model that enables the establishment of axiomism. It is called Consciousness Model. With the axiomism, a Three-dimension-like Space Model (TDM, called hereinafter) is defined. Axiomatically, TDM is replaced by programs called Tense Control Function and Pallet Function, three kinds of program called Signification Vector, and four kinds of program called Action Vector.

By these nine kinds of program, it is re-defined. It is the entity of SF. Signification Vector and Action Vector are defined on the basis of Predicate Structure introduced from this axiomism. Signification Vector, in particular, concludes to be a mutually-independent one-variable proposition.

Three kinds of world with Unit [1] as its constituent are produced, and they assume a role to establish three-dimensional coordinates system. In the space of the three-dimensional coordinates system, another two kinds of world are placed. These two kinds of world are also a world with Unit as its constituent, same as the aforementioned three kinds of world. The two worlds establish the whole with a relation of a set and its complementary set. The two kinds of world are called denotative world (A^+) and connotative world (A^-). This is an outline of the structure of TDM. The thinking of TDM is described further in the section 2. 3.

In Consciousness Model, if Unique Unit [1] materialises in the denotative world and Critical Unit [1] in the connotative world, a corresponding relation of the two kinds of world shall be established. The state is a structure of the materialisation of an intent. The state is dynamic, and for this reason, we cannot grasp the whole of it. However, this subject targets to grasp an intent. To attain it, it is indispensable to grasp an intent statically and unitarily although it materialises dynamically .

For that sake, this relation is not obtained in the three-dimensional space of TDM, but it is substituted into the three kinds of world that can establish static state. The three-dimensional space of TDM is a stage for the establishment of dynamic appearance.

For this purpose, by using the concept of the Unique Unit and Critical Unit, the Unit belonging to the three kinds of world is recreated into a static structure that can perform the same role as they do. This procedure must be implemented axiomatically in the same manner as the concept of Unique Unit and Critical Unit was defined axiomatically. The structure of the Unit of the re-obtained three kinds of world is Predicate Structure.

The materialisation of the two kinds of world's corresponding relation in TDM means that the boundary point of the complementary set relation of the two kinds of world can be obtained in TDM, and it is synonymous with that the coordinates' origin of TDM can be obtained. If this relation is expressed by SF and executed on computer, the boundary point of the two kinds of world and its locus can be obtained.

The reflection of the appearance is the materialisation of systematisation of significant data mentioned in the beginning. It is nothing but a requirement itself that reflects an intent to materialise.

The above-mentioned is a relationship of consciousness and requirement that is defined by the axiomism of Consciousness Model. That is, SF is different from the traditional program that is logically described in advance based on the knowledge and experience related to an intent, with view to its structure and nature.

2 Scenario Function

If the traditional method to determine program is observed, it is thought that a requirement is attempted to grasp by the method approaching from the connotative world to the denotative world. In contrast, SF shall produce a requirement by the method different from it. Hereupon, centering on that matter, SF is explained.

2.1 The Origin of Scenario Function

TDM is a world realised only by theorem. The theorem is delivered from the hypothesis of Consciousness Model. A concept of the factor to establish the world is the Unit, and another concept to establish the world's corresponding relation defines the TDM. Fig. 1 shows the outline of these relations, where the concept of synchronicity plays an important role . TDM is realisation of synchronicity under the hypotheses of this theory , SF is realisation of TDM in programming language, and Conscious Model provides axiomism that gives a theoretical base for the establishment of TDM and SF.

2.2 Meaning of Scenario Function

In the hypothesis of this subject, as already described, an intent is only one set. Further to mention, an intent possesses density. For example, the number of nouns belonging to therein. The necessary and sufficient number of nouns to materialise an intent means all nouns. It is designated by *U*, and its partial density by *A*. The definition of *U* is possible in the connotative world that represents a conceptual world. However, it is impossible in the denotative world that represents a world to be actualised. Therefore, the definition of an intent is done in the connotative world and it is expressed by $T^{I}(U)$.

Same as the case of *U*, the definition of the density of the whole *A* is possible in the connotative world but impossible in the denotative world. Therefore, the definition of the whole *A* is done in the connotative world and it is expressed by $T^{I}(A)$.

It is the partial density of *A* whose definition is established in the both connotative world and denotative world, and if it is designated by A_i , the definition of the partial density of *A* is $T^{I}(A_i)$, following after the form of the preceding definition. Its definition in the denotative world is as follows, and it is the definition of SF itself:

$$\Phi[(\Phi 4,\tau)[\{L4,i\}+\{O4,r_{\alpha}\}+\{S4,r_{\beta}\}+R4]+ (\Phi 2,\tau)[\{L2,j\}+\{I2,r_{\gamma}\}+R2]+(\Phi 3,\tau)[\{L3,i\}+R3]] \quad (1)$$

Items composing SF are explained in the following sections. One SF or a combination of plural SF's can express $T^{I}(A_i)$, and a SF is called Basic Structure in either case.

Only limited to the case of A_i , the connotative world and the denotative world become synonymous. That is, SF can be established. SF determines necessary commands universally and establishes the line-up order of those commands after determine it universally. The execution of $T^{I}(A_i)$ means an action to repeat it. That

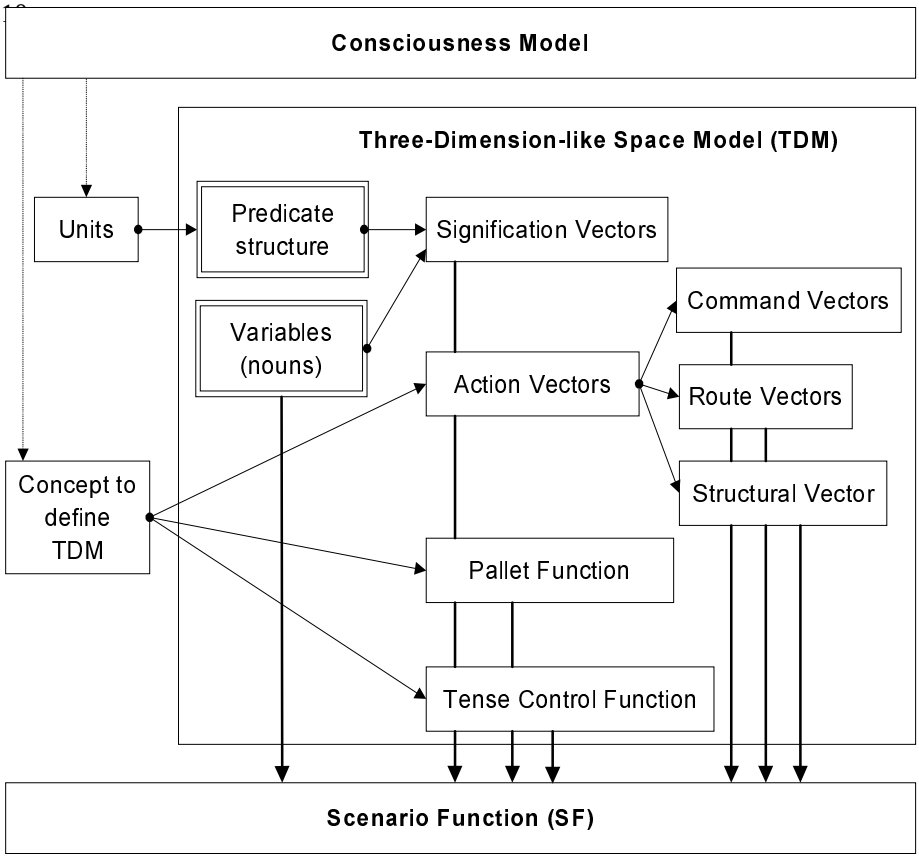

Fig. 1. The Origin of Scenario Function. A dotted arrow (⋯▸) represents theorems to create synchronicity. An arrow with a starting point (●⟶) represents realisation of synchronicity in TDM. A thick arrow (⟶) represents realisation of an intent in programming language.

state is expressed by $T^n\ (A_i)$. The subscript n is an index of the indication of its repetition. By the execution, it converges into the state of *A*. That is, it establishes a relation of grasping *A*.

Speaking from this subject's standpoint, the traditional program adopts a method targeting a program which reflects A directly. As a result, the tendency of logical definition appears strongly. The problem of productivity occurs thereabouts. The difference of these approaches explains why our methodology can solve problems that traditional methods have caused. For example, the final structure of software can be established universally by SF, the upper-stream work is simplified as a result. This is clear through the software development we have done for our clients [2].

The state of the memory after executing the traditional program reflecting an intent *A*, and that of $T^n\ (A_i)$ become identical. By using this relation, algorithm can be determined, which autonomously converts traditional programs into SF. By using the algorithm, a tool has been developed which automatically converts a traditional

program in Assembler into SF in COBOL and a traditional program in COBOL into SF in COBOL.

This conversion algorithm can be determined easily as for any languages. Also, algorithm to convert SF into programs of a traditional type can be determined, too. The conversion algorithm does not need to be reflected on the whole SF. For example, algorithm for the matrix calculation is just to be placed in the 2nd box of the Signification Vector on the Pallet *W04*. Of course, it is possible to develop it in the whole SF. Pallets are explained in the following sections.

2.3 Three-Dimension-like Space Model

A world reflecting an intent of necessary and sufficient density is a denotative world, and a world reflecting an intent of insufficient density is a connotative world. The denotative world is outside of our re-cognition. In TDM shown as Fig. 2, the denotative world reflecting an intent *A* is expressed by A^+, and its element is denotative Unit. The connotative world reflecting an intent A is expressed by A^-, and its element is connotative Unit. As already explained, the two kinds of worlds are placed in the three-dimensional space of TDM.

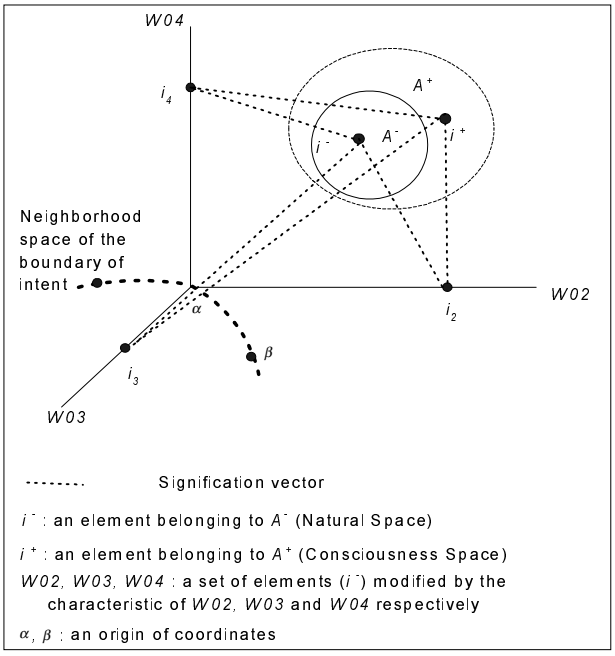

Fig. 2. Three-Dimension-like Space Model

The Unit is defined as a relation of a subset of logical atoms [1] that produces an existence to one logical atom that corresponds to the subset. The logical atom is a concept created in the Consciousness Model. Unit is distinguished into either case that the equivalent atom belongs to its subset or that it doesn't. The state of the former equivalent atom gives a whole nature to its subset, and the latter gives a partial nature to its subset. For this, the former Unit is called denotative Unit, and the latter connotative Unit.

The materialisation of an intent *A* is realised upon the occurrence of common sharing of the equivalent atom of the connotative Unit and denotative Unit of the two kinds of world A^+ and A^- , that reflect the intent. That particular Unit is called Critical Unit of the connotative Unit, and Unique Unit of the denotative Unit. These Units can

share the equivalent atom. As already described, the Predicate Structure is delivered from the concept of these Units. It is one of the important conclusions of this subject, but the detail of its delivery is out of the scope of this paper.

TDM establishes a corresponding relation of the two kinds of world A^+ and A^-, by using the three kinds of world with Signification Vector defined based on Predicate Structure as its element. In other words, it is a structure to determine the equivalent atom that makes the correspondence between the two kinds of world A^+ and A^- . It is inevitable that the determined equivalent atom is positioned as the origin of the TDM's coordinates system.

The reason why TDM becomes a three-dimensional-like structure is that there are three kinds of world intervene between the connotative world and the denotative world in Consciousness Model.

The three kinds of world constitute one world which performs the role to correspond the two kinds of world and is the connotative world as A^-. Each of the three kinds of world possesses unique nature. *W04*, *W02* and *W03* in Fig 2 represent the three kinds of world, and the Unit belonging to those worlds reflects nature of its own world. That is, the Unit becomes Signification Vector that reflects the nature of its own world and becomes an element of the three kinds of world. *W04*, *W02* and *W03* are called Pallet. A Predicate Structure to make correspondence of A^+ to A^- in TDM is derived from theorems of this methodology. Propositions, which are also called Signification Vector, are determined on the basis of the Predicate Structure, and by executing them on the computer a correspondence between A^+ to A^- is established. The correspondence is an intent, whereas integration of the TRUE state of the propositions is what we call a requirement reflecting the intent. The meaning of the TRUE state of a proposition is described in the following section.

2.4 Signification Vector

To reflect an intent in natural language is equal to collect subsets from among the set of words existing innumerably and to assign a sequential order to the words belonging to the subsets. This is a work to make the correspondence of the infinite number to the finite number, and it causes problems from algorithm viewpoint. Therefore, this action ought to originally become polysemous (i.e., a state of multiplicity of meaning). However, we have no means but to manifest it in one and only way. As a result, it determines self-will. This matter is enough to relate that, except becoming accustomed to it, there is no true way of making it public. Consequently, a self-will is a problem deep-rooted in a way of our recognition, and it is thought that the action of verb has a deep relation to it. Therefore, we are to obtain Predicate Structure, thereby excluding verb.

As shown in Fig. 3, one piece of Predicate Structure is comprised of seven kinds of rule. Nouns belonging to a requirement are defined as Signification Vector respectively by these seven kinds of rule. The rule of the 2nd box can be defined by the information of user's definition, and the rest six kinds of rule can be defined with determinism by the universality of Predicate Structure as well as TDM. Pallet is three kinds (*W04*, *W02*, *W03*), and Signification Vector belonging to each Pallet is defined in accordance with

the Pallet's nature. Signification Vectors place on *W04, W02* and *W03* are respectively denoted as *{L4,i}, {L2,j}, {L3,i}*. The seven kinds of rule are explained in the following paragraphs.

The First Box. The 1st box of Predicate Structure is a rule to make judgement if the 2nd box of the same Signification Vector must be executed or not. Signification Vector belonging to *W04, {L4,i},* checks the 4th box of *W03* Signification Vector having the same noun as its own, *{L3,i}*. If a value is set therein, the *W03* Signification Vector is TRUE. In that instance, the *{L4,i}*'s 2nd box rule is executed. If a value is not set the 4th box of *{L3,i},* the *{L3,i}* is FALSE. In that instance, the *{L4,i}* ends its role.

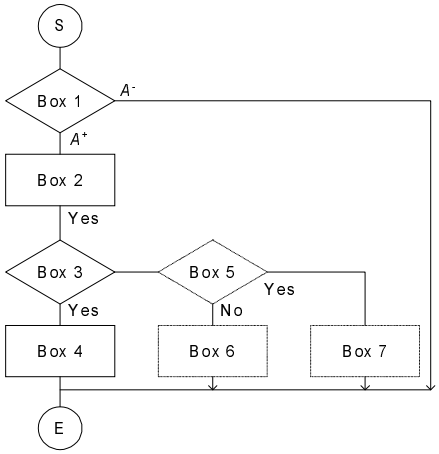

Fig. 3. Predicate Structure

Signification Vector belonging to *W02* and *W03* checks the 4th box of their own Signification Vector. If a value is set therein, the Signification Vector is TRUE. In that instance, it is not necessary to execute the 2nd box rule, so this Signification Vector ends its role.

The Second Box. The rule of the 2nd box of Predicate Structure is to tentatively set a value to be set to the memory area of the 4th box. For this sake, a temporary memory area of every noun is prepared. The 2nd box rule of *{L4,i}* is to operate or duplicate the value corresponding to the variable (noun) of the Signification Vector. An instruction to be filled in this box is given by the user as a requirement although it is merely a part of an actual requirement.

A variable that holds a value to be duplicated or used for calculation of another noun is called the 2nd coordinate. A noun of the 2nd coordinate is not always allowed to exist in any Basic Structure in the Process Route Diagram. The relation between the Pallets or the Basic Structures in which the variable and the 2nd coordinate are placed should guarantee synchronicity. For example, a noun in the Pallet or the Basic Structure placed in the odd ordinal numbers cannot become the 2nd coordinate noun. This is a theorem of TDM. However, when a noun in such Pallet or Basic Structure needs to be used as the 2nd coordinate, a memory area of the noun is prepared in advance on the Pallet synchronising with that of the variable so that it can become the 2nd coordinate. A noun becoming the 2nd coordinate in this manner is called Boundary Word . The 4th box rule of the Signification Vector of a Boundary Word sets a value not only in its memory area but also in the memory area to be used as Boundary Word. Process Route Diagram is explained in the section 2.7.

The 2nd box rule of *{L2,j}* is to duplicate a value corresponding to the variable (noun) from the input memory area (buffer) into its temporary memory area. The 2nd box rule of *{L3,i}* is to tentatively prepare the executing condition which is checked by the 1st box rule of *{L4,i}* of the same variable. It is a rule of IF judgement of conventional ideas . The judgement result is set in the temporary memory area of the 2nd box.

The Third Box. The rule of the 3rd box of Predicate Structure is to implement significance judgement of the value obtained by the 2nd box rule. The 3rd box rule of Signification Vector is the same regardless of the nature of the Pallet to which it belongs. That is, it is a rule to judge if the value set in the temporary memory area of the 2nd box is significant or not. If significant, the process proceeds to the 4th box rule. If not significant, the process proceeds to the 5th box. When proceeding to the 5th box, this Signification Vector is FALSE as a proposition.

The Fourth box. The rule of the 4th box of Predicate Structure is a rule to duplicate the value in the temporary area of the 2nd box into the actual area of the variable. The 4th box rules of *{L4,i}*, *{L2,j}* and *{L3,i}* are the same . It is a rule to duplicate the value set in the temporary memory area of the 2nd box into the actual memory area. When plural calculation expressions exist conditionally for one noun to obtain a TRUE value, the number of *W03* Signification Vector taking the noun as variable is only one, whereas plural *W04* Signification Vectors are required. Such noun is called Equivalent Word.

The fifth, sixth and seventh boxes. The rules of the 5th, 6th and 7th boxes of Predicate Structure are not delivered axiomatically. They are prepared to match TDM to the mode of the current computer. Signification Vectors of any Pallet have the same rule.

SF consists of Tense Control Function (Φ), Pallet Function ((Φ4,τ), (Φ2,τ), (Φ3, τ)), Signification Vector (*{L4,i}*, *{L2,j}*, *{L3,i}*) and Action Vector (*{O4,*r_α*}*, *{S4,*r_β*}*, *{I2,*r_γ*}*, *R4*, *R2*, *R3*). Signification Vector and Action Vector are generically called Tense Control Vector. Tense Control Function appoints Pallet to be executed. Pallet Function executes all the Tense Control Vectors on the Pallet as a group and iterates until the state transition of the Pallet does not occur any longer. State transition is judged if the state of all the Tense Control Vectors on the Pallets have become TRUE or any changes of TRUE/FALSE state is not expected. This judgement is done by the 5th box rule. When the state transition of the Pallet is not expected, Route Vector declares the next Pallet, and Tense Control Function appoints it. In this manner, SF is executed in repetition, thereby proliferating the TRUE state. This is a complementary action. The execution of the 5th box rule is limited to the only case the Signification Vector is FALSE. When the Signification Vector is already TRUE, the role ends in the 1st box. Signification Vector declares execution of itself again if the Signification Vector is not TRUE. It is the 7th box rule. If Signification Vector is not expected to become TRUE, the Signification Vector in the FALSE state autonomously declares the refusal of re-execution. It is the 6th box rule. The relation of the request for re-execution and the refusal is complementary. The result of the 6th box is checked by the 1st box rule. The result of the 7th box is checked by the Pallet Function. A sample program of the Signification Vector appears on our web page [3].

2.5 Complementary Action

The complementary action is referred in the previous section, and a little more meaning of it is added herein. When SF is executed, a relation indicated by arrows in Fig. 4 is produced. This relation is autonomously produced among all Tense Control Vectors. This is the appearance of the complementary action. Whereas a traditional program is

to be developed to assure its sequential execution, SF is free from an idea of a sequence of execution owing to the complementary action. The complementary action is realised because the Predicate Structure creates the structure of memory areas deterministically as shown in Fig. 5. With understanding of the complementary action and the meaning of the 1st box and 3rd boxes, it is clear that the logical verification of SF is not required at all.

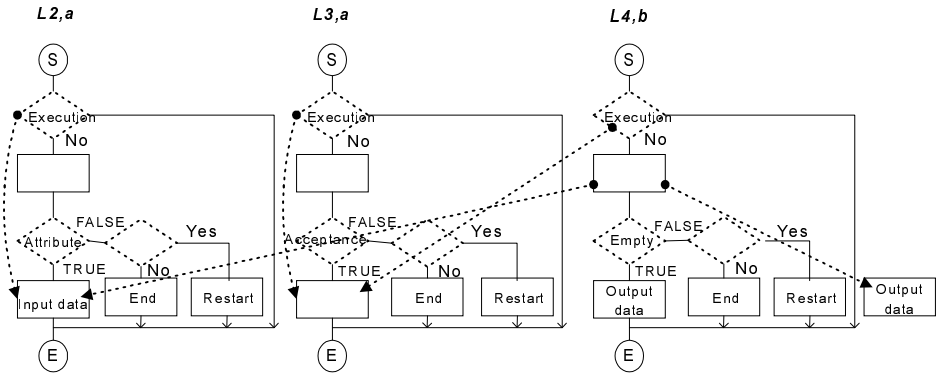

Fig. 4. Complementary Action. The 2nd box of *L2,a* acquires input data, the 2nd box of *L3,a* acquires complementary conditions, and the 2nd box of *L4,b* generates output data.

Pallet	Type of vectors (1)	Tense control vectors (1)	Self word area				Control box area				
			Box 2	Box 4	Box 6	Box 7	Status	Current access key	Previous access key	EOF	MSGFLG
W04	SV	1. SV for output word state control vector	●(2)	●	●	●					
	AV	2. Output vector & duplication vector	●		●	●	●	◊(3)	◊	●	●
		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		●							

Fig. 5. Tense Control Vectors and Memory Areas. (1) SV and AV denote the signification vector and the action vector respectively. (2) A dot denotes that the memory area is required. (3) A diamond denotes that the memory area is required in certain cases. (4) A shaded box indicates that the memory area is not necessary.

2.6 Action Vector

The structure of Action Vector is Predicate Structure basically, and it is diverted for the use of Action Vector. The Signification Vector is defined for every noun, but the Action Vector is defined for every set of nouns made into a unit. The so-called logical record is a set of nouns made into a unit. If Action Vector is also regarded as a proposition, the TRUE/FALSE relation is determined by the state of the 4th box, same as the case of Signification Vector. Action Vectors are categorised into four kinds as Input, Output, Structural and Route Vectors.

If the environment for execution is defined, the Input Vector and the Output Vector are generated from that information. If logical records and nouns are defined, the Structural Vector and the Route Vector are generated from that information. Sample programs of the Action Vector are shown on our web page [3]. The roles of respective Action Vectors are as follows:

Input Vector. This Action Vector, denoted as *{I2,r_γ}*, duplicates a value in a physical record into its input memory area called logical record *(r_γ)*. This Action Vector is defined for every input logical record *(r_γ)* and placed in *W02*. In case that the same Input Vector is defined in plural Basic Structures, if it becomes TRUE on one of the Basic Structures, the Input Vector on all the Basic Structures become TRUE.

Output Vector. This Action Vector, denoted as *{O4,r_α}*, duplicates a value in the W04's Pallet Word memory area into the output memory area called logical record, *(r_α)* and then outputs it on an external device. This Action Vector is defined for every output logical record *(r_α)* and placed in *W04*.

Structural Vector. This Action Vector is denoted as *{S4,r_β }*. When Output Vector becomes TRUE, that is, output has been done, Structural Vector initialises every memory area of a Basic Structure with which the Output Vector is concerned. Structural Vector is defined for each memory area, r_β. A memory area to be regarded as a unit to be initialised is as follows:

- on *W02*, an input memory area, r g , and a *W02* Pallet Word area;
- on *W03*, a *W03* Pallet Word area;
- on *W04*, an input memory area, r a , and a *W04* Pallet Word area.

A Pallet Word area is a memory area defined by Signification Vector.

Route Vector. Tense Control Function controls Pallet transition and this Vector generates information for the transition. Even if there are a plural number of Pallets as a destination, one Route Vector is sufficient for a Pallet. Route Vectors on *W04, W02* and *W03* are denoted as *R4, R2* and *R3* respectively. The classification of Route is the following seven kinds.

- Route 42: It is a route to proceed from*W04* to *W02* in Basic Structure (τ), that is, $R4_\tau$: *W04*, $_\tau \rightarrow W02_\tau$
- Route 23: It is a route to proceed from*W02* to *W03* in Basic Structure (τ), that is, $R2_\tau$: $W02_\tau \rightarrow W03_\tau$
- Route 30: It is a route to stop operation after *W03* in Basic Structure (τ), that is, $R3_\tau$: $W03_\tau \rightarrow$End
- Recursive Route: It is a route to return to *W04* from *W03* in Basic Structure (τ), that is, $R3_\tau$: $W03_\tau \rightarrow W04_\tau$

- Continuous Route: It is a route to proceed from *W03* to *W04* of one of the succeeding Basic Structures, that is, $W03 \rightarrow W04_{\text{next}}$
- Duplex Route: It is a route to return from *W03* to *W03* of the previous Basic Structure, that is, $W03 \rightarrow W03_{\text{previous}}$
- Multiplex Route: It is a route to return from *W03* to *W04* of the preceding Basic Structure when the two Basic Structures are not adjoining each other, that is, $W03 \rightarrow W04_{\text{preceding}}$

2.7 Process Route Diagram

The plural number of SF's are connected each other using the information provided by Route Vectors, and the connected SF's establish a structure that expresses an intent. The structure is called Synchronous Structure and the diagram of the connected SF's , shown in Fig. 6, is called Process Route Diagram, PRD.

Information on screen transition, for example, corresponds to PRD although screen transition should be defined beforehand. PRD, on the other hand, can be drawn theoretically owing to the structural characteristics of SF and does not need to be defined in advance.

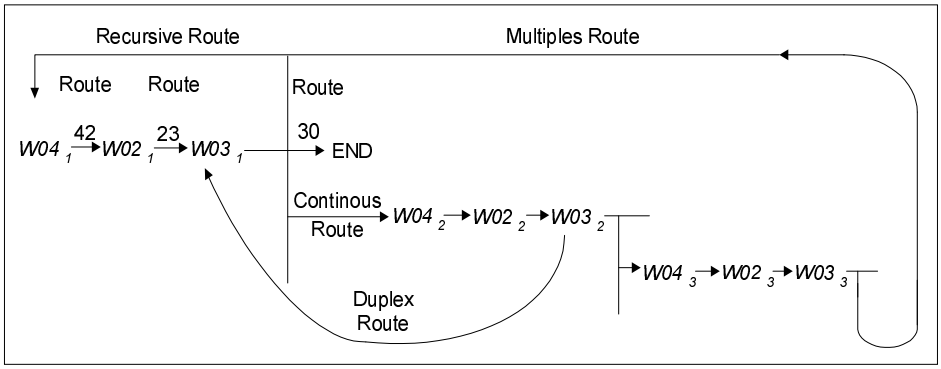

Fig. 6. Process Route Diagram. Arrow are routes defined by Route Vectors.

2.8 The Structure of Pallet Area

Memory areas are assigned on specific Pallets as follows. A memory area for an input logical record is in *W02*, that for an output logical record is in *W04*, that for a word of input attribute is in *W02* and that for a word of output attribute is in *W04* and *W03*. These relations of the Tense Control Vector to the memory areas are based on determinism. Fig. 5 shows the placement of memory areas.

3 Conclusion

We expect an intent to be manifested by using words of the natural language in a certain sequential order. However, it is not certain in what manner an intent materialises, and it is not certain how the language concerns itself with it.

The purpose of this study is found in realising universality in the requirement of software development. That is, supposing that the requirement originates in an intent, and in obtaining its origin, an observation is made, by obtaining the rule of theorem materialising therein, for establishing universality in the requirement based on the rule of theorem. It is the proposition of this study.

A model is built, which realises axiomism so as to enable to define the origin of an intent axiomatically. It is the Consciousness Model. In this theory, however, the Consciousness Model and its axiomism are not explained. From this model, the Three-dimension-like Space Model and Predicate Structure are delivered. That is, SF is what represents the origin of the hypothesised intent. By executing it, a relationship materialises as if an intent which creates a requirement can be obtained. In other words, it can establish the complementary action of Signification Vectors in the set of Signification Vectors. Because of this, the sequence of the requirement definition can be removed. This means the requirement can be grasped resultantly without being deeply concerned with the requirement. That is, SF has succeeded in realising the relation in which a requirement is delivered from an intent.

- SF uses nouns belonging to a requirement as a factor to materialise as an intent.
- SF converts the traditional chaotic program structure into a universal structure.
- SF reduces traditional works such as designing, production and verification.
- SF can change a way of recognising software from conventional to revolutionary.
- More than 90% of the total work volume of the traditional software development can be automated by SF.

All of these above-mentioned may as well qualify SF to be the most innovative methodology of software development. In Japan, SF has already been used for actual systems of small to large scales, and innovative results have been proven in the productivity and maintainability. Thus, the methodology is highly anticipated to be used in the world .

The idea presented here is a new definition of software deeply influenced with the works of Spinoza, Leibniz and Wittgenstein.

References

1. Negoro, Fumio : Principle of Lyee Software. Proceedings of 2000 International Conference on Information Society in the 21 st Century, IS2000 and the University of Aizu (2000), 441-446.

2. Record of System Development. At http://www.lyee.co.jp.

3. Lyee Scenario Function-sample programs. At http://www.lyee.co.jp.