The Predicate Structure to Represent the Intention for Software

Fumio Negoro
The Institute of Computer Based Software Methodology and Technology
f-negoro@lyee.co.jp

Abstract

The requirement in this study is a sentence written in a natural language. This is a methodology to determine programs by mechanically extracting words from the requirement, where the words are nouns in the sentence. The program is called Scenario Function (SF). SF is a set of propositions. The proposition is determined with the predicate structure that takes a noun as its variable. The predicate structure has been derived from this study. The nouns used as variables are clues to realize on SF the meaning and the functionality contained in the sentence. The requirement that cannot be expressed only with the nouns is realized by the complementary action of the propositions, which is realized when SF is executed on computer. In other words, the most of the work required in the conventional ways of software development such as requirement definition, logical design, program design and tests, can be eliminated and is no longer the primary part of software development with this methodology. As a result, a human work in software development can be changed into a work with less burdens.

1. The Core of non-engineering hypothesis

In this section, the core of the hypothesis is briefly mentioned. We have studies what an intention is. The conclusion is that an intention is made when a correspondence between two worlds is made and that the two worlds are materialized with a set of particular elements. Time has a variety of velocities that are independent elements. The elements are nonexistent to us. These elements are the elements of the two worlds. The separation of the worlds is made based on the difference of the time velocity. One of the worlds is called the connotational world and the other is the denotational world. If a correspondence is made between the two worlds, that state becomes an intention. At that moment, the elements in the connotational world are objectified.

The objectified element cannot return to the state that it has been objectified. The behavior of the objectified elements is the whole of the object of our cognition. However, we are unable to cognize the whole.

Sentences, requirements and natural languages consist of the objects; therefore, they are never complete as they cannot represent the whole. The model to realize this hypothesis is the Consciousness Model, where the above hypothesis is made into axioms. [1]

2. TDM and SF

According to the axioms, a requirement is a set of objects reflecting the intention. Therefore, it is not that we are able determine a requirement autonomously. All depends upon the intention. The purpose of this study is to find a language to represent the intention. However, in this paper, a methodology to represent the intention in a programming language is presented.

The scheme is to be realized, which simultaneously satisfies a process of reversing a requirement that comes to someone's mind at a certain moment to the world where the intention has been made, and a process of enabling the creation of a requirement that fully reflects the intention. A structure is to be realized, which leaves the above processes to an action of computer, manifests a requirement into the memory of computer, and makes the intention in the mind of the computer users. The scheme is defined as Three-Dimension-like Space Model (TDM) by the axiomism, and it is expressed in a programming language so as to leave it to a computer. This is the Scenario Function (SF). [2] [3]

TDM is a static state of the aforementioned scheme, and SF is the same. SF changes a static state of itself into a dynamic structure on computer. TDM and SF on execution realize the hypothesis of the Consciousness Model. The two worlds are placed in the space in TDM, and nouns are placed on the three coordinates of TDM. The rules of placement of the nouns are determined according to the nature of each coordinate. The nature of the coordinate is derived from the axioms. There exist the particular elements in the two separate spaces in the TDM. Those elements are not cognizable.

However, a certain correspondence is made between the nouns on the coordinates and the elements. The correspondence is also defined by the axioms. This correspondence is the predicate structure. The predicate structure is a structure to redefine the nouns as the hypothesized elements. When this predicate structure becomes propositions, they are called Signification Vectors.

The origin of TDM is determined when the intention is made and objectification of elements is done. In other words, the intention means a closed boundary line of a certain domain according to our definition, and the inside of the domain corresponds to the connotational world and the outside corresponds to the denotational world. A set of the origins of TDM corresponds to the boundary line. Therefore, execution of SF simultaneously defines the origins and a set of them.

The meaning of execution of SF can be explained in another way. If the number of object of someone's requirement is *N*, SF is determined by *N*, whereas when SF is executed, N^N objects are created by the complementary action as a possible number of the objects. In the conventional ways, only *N* objects can be turned into programs and the *N* object must be determined logically beforehand. On the contrary, with this methodology, 3*N* Signification Vectors are determined deterministically and N^N objects are realized.

3. Definition of SF

SF is a structure to realize TDM by using computer. SF can be defined based on a set of nouns that manifest a certain psychological state. A screen and a file are a set of nouns. They become a basis to define SF. To develop software concludes to define SF. In the following, a general structure of SF is defined.[1]

$$SF_a = \Phi[\Phi 4(\{L4(j)\}+\{O4(r)\}+\{S4(a)\}+R4) + \Phi 2(\{L2(i)\}+\{I2(r)\}+R2) + \Phi 3(\{L3(j)\}+\{R3R, R3D, R3M, R3C\})]_a$$

Herein, *{L4(j)}*, *{O4(r)}*, *{S4(a)}*, *R4*, *{L2(i)}*, *{I2(r)}*, *R2*, *{L3(j)}* and *{R3R, R3D, R3M, R3C}* are generally called Tense Control Vector. There are 12 kinds in all. *L4(j)*, *L2(i)* and *L3(j)* are called Signification Vectors, whereas the rest of the Tense Control Vectors are generally called Action Vectors. *O4(r)*, *I2(r)* and *S4(a)* are Output Vector, Input Vector and Structural Vector respectively. *R4*, *R2*, *R3R*, *R3D*, *R3M* and *R3C* are Routing Vectors. The definition and role of each Tense Control Vector is explained below.

The Signification Vector is defined for every noun. The *j* and *i* respectively represent output and input attribute of a noun. The role of *L2(i)* is to duplicate information from an input memory area to its own memory area. The role of *L4(j)* is to create information to be stored in its own memory area by complementary action. The role of *L3(j)* is to store in its memory area an execute directive of Signification Vector *L4(j)*.

The *O4(r)* is defined for every output memory area, *r*. The role is to output data in the output memory area to an output physical unit. The output attribute noun, *j*, is a noun belonging to the output memory area. The *I2(r)* is defined for every input memory area, *r*. The role is to obtain data from an input physical unit into its input memory area. The input attribute noun, *i*, is a noun belonging to the input memory area. The *S4(a)* is defined for every memory area, *a*, requiring initialization. The role is to initialize the area. *R4*, *R2*, *R3R*, *R3D*, *R3M* and *R3C* specify Pallet Function that is executed next.

The Tense Control Vectors are bundled by the three kinds of Pallet Functions, *Φ4*, *Φ2* and *Φ3*, as follows: *Φ4({L4(j)}+{O4(r)}+{S4(a)}+R4)*, *Φ2({L2(i)}+{I2(r)}+R2)*, and *Φ3({L3(j)}+{R3R, R3D, R3M, R3C}*. The symbol *{}* is a set symbol to bundle Tense Control Vectors, and the + is a symbol to indicate the execution order. For example, Tense Control Vectors bundled in a Pallet Function, *Φ4*, are placed in the order of *L4(j)*, *O4(r)*, *S4(a)* and *R4*, and they operate in this placement order. This is expressed by *Φ4({L4(j)}+{O4(r)}+{S4(a)}+R4)*. In a set of kind-wise Tense Control Vectors, a symbol to indicate execution order is not contained. For example, the set *L4(j)* means that the way of the placement of the Signification Vectors of *L4(j)* is free, and it operates in the free-placement order upon execution. A set of Tense Control Vectors bundled by the Pallet Function is generally called Pallet. The Pallets to be bundled by *Φ4*, *Φ2* and *Φ3* are respectively expressed as *W04*, *W02* and *W03*.

The Tense Control Function, *Φ*, assigns execution-right to the next-to-operate Pallet Function. A Pallet Function having been given the execution-right gives the execution-right to all Tense Control Vectors belonging to the self. Once Tense Control Vectors are thoroughly executed, the execution-right is returned to the Pallet Function. The Pallet Function judges whether its Pallet needs to be re-executed or not. If the re-execution must be repeated, the execution-right is again given to all Tense Control Vectors belonging to the self. If otherwise, the execution-right is returned to the Tense Control Function. Thereafter, the Pallet Functions make execution in the order of *Φ4*, *Φ2*, *Φ3*, under the Tense Control Function. It is defined in the format of *Φ(Φ4+Φ2+Φ3)*. This is an expression of the SF definition. The execution of SF and the Tense Control Vectors are described the sections 6 and7.

[1] Programs of the SF are shown as Lyee Scenario Function – sample programs at http://www.lyee.co.jp.

4. Predicate structure

If an element of the denotational world that corresponds to an element of the connotational world exists, the element is made exteriorized because it can actualize as a real existence with view to the hypothesis. However, if it has already been actualized, it is not necessary to actualize in repetition. Also, if there is no corresponding element in the connotational world, an element of the denotational world cannot be actualized. The actualization means to make the self actualize by utilizing other element that has already been made into a real existence. This is a complementary action. However, the complementary action must be what reflects an intention. It is implemented in accordance with the providence of actualization. This means a state in which the mutual relation concerned with the complementary action establishes Synchronous Structure. Through this, the actualization is recognized. If otherwise, the complementary action must be done over again, or, stop doing it. All mentioned above is represented as the predicate structure shown in Fig. 1. The predicate structure consists of seven boxes and each box contains a specific rule. The definition of all operators that are shown in the definition of SF is effected by the determination of the seven kinds of rules of the predicate structure. The 1st rule is contained in the first box, and so on. Each of the 2nd, 4th, 6th and 7th boxes has a memory area where the result of execution of the rule is to be stored. Those memory areas are called the 2nd, 4th, 6th and 7th areas respectively. The predicate structure becomes a Signification Vector when a noun is given to it as its variable, whereas the predicate structure becomes an Action Vector when it takes a plural number of nouns as its variable and performs a specific role.

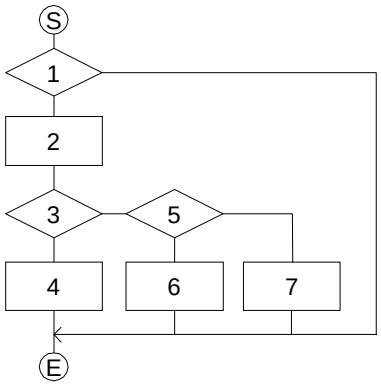

Figure 1. Predicate structure

The six kinds of rules except the 2nd rule are autonomously determined based on the definition information of the 2nd rule, so the definition action is a work for the 2nd rule only practically. The 2nd rule is already fixed by what must be regulated by each operator, so SF can be defined simply by implementing the regulated work. This work is already done with a tool. Associated work elements required to execute SF on computer, for example, logical and physical definitions of a screen must be fulfilled separately from the above work as a natural course of procedure. A Signification Vector is a proposition because it is defined by taking one piece of noun as its variable, and an Action Vector is a function which is defined as a functionality that takes plural nouns as variable.

The Signification Vector is true as a proposition if a value is being set in its 4th area. The complementary action of Signification Vectors materializes as a relation of propositions that become true. The Input Vector materializes as a function if information can be obtained in its 4th area from an input physical unit. The Output Vector materializes as a function if a flag indicating success of an output action is set into its 4th area. The Routing Vector materializes as a function if the next-to-execute Pallet is appointed in its 4th area.

5. Synchronous structure

In this study, a relation to establish an intention and a requirement of its reflection concurrently is called Synchronous Structure. That is, it is a world of TDM and SF. The synchronous range means the range of an intention to materialize as well as the range of a requirement of its reflection. A plural number of SFs can be connected each other. In liaison with one another upon execution, plural SFs realize expansion or reduction of the synchronous range. Same as SF can be defined statically, a state providing the liaison can also be defined statically by using SFs as elements. The diagram to show the liaison of SFs is called Process Route Diagram, hereinafter denoted as PRD. SF can be defined as a set of nouns which are unitized as a requirement, for example, by taking one screen and a set of nouns belonging thereto as a unit. Therefore, a SF can be defined in correspondence to each of the screens in the screen transition. By taking SFs as elements and based on the relation of the screen transition, PRD can be defined. Owing to the SF's structural features, PRD can be drawn through mechanical procedure if only the information of individual screens can be defined. Therefore, if this procedure is mechanized by a tool, a work to define the screen transition information can be substituted by the tool.

A rule of liaison of plural pieces of SF is four kinds. It is reflected as a definition of Routing Vectors, *R3R*, *R3D*, *R3M* and *R3C*, belonging to *W03*.

6. Execution of SF

Once a Pallet's Tense Control Vectors are executed thoroughly, the status of the Pallet is recorded. Recorded is a status of just before and current. The status of a Pallet means a set of status of the 4th area (true/false as a proposition), the 6th area (refusal flag) and the 7th area (restart flag) of all the Tense Control Vectors belonging to the Pallet. These areas are generally called Pallet Area in particular. Routing Vectors belonging to *W04* and *W02*, i.e. *R4* and *R2,* compare the just-before and current status of the self Pallet, and if there is a state transition, the execution of the self is terminated. If there is no state transition, the Pallet to be executed is appointed in the self's 4th area. *R4* appoints *W02* of the same SF, *R2* appoints *W03* of the same SF.

Pallet Functions of *W04* and *W02*, i.e. *Φ4* and *Φ2*, implement check on the 4th area of *R4* and *R2* belonging to the self Pallet, once all Tense Control Vectors belonging to the self Pallet are executed thoroughly. Then, if the next-to-execute Pallet has been appointed thereabouts, the Pallet Function transfers execution-right to the Tense Control Function, *Φ*. Based on the information, the Tense Control Function transfers execution-right to the next-to-execute Pallet Function. If the next-to-execute Pallet has not been appointed thereabouts, the Pallet Function gives execution-right to the Tense Control Vectors belonging to the self and re-executes from the beginning. There are four kinds of Pallets to be appointed as the next-to-execute Pallet by the Routing Vector belonging to *W03*. For this reason, four kinds of Routing Vectors, *R3R*, *R3D*, *R3M* and *R3C*, are defined on *W03*. Details of Routing Vectors are described later.

The status of Pallets symbolizing the appearance of the SF execution is a result of executing Signification Vectors. It is a result of the complementary action to materialize among Signification Vectors to be executed. Therefore, the presence of the state transition of Pallet Areas is a condition to determine whether SF is to be executed or not. That is, if a state transition occurs, it indicates that there is still a possibility of materializing complementary action in the SF, and if no state transition, it indicates that a complementary action does not materialize.

Based on the data which is obtained with the Input Vector, SF establishes the complementary action autonomously among *W04* Signification Vectors. Then the Output Vector determines the state of the result of the complementary action so as to establish another complementary action with us. That is, the Output Vector performs a role to manifest (output) a result of the complementary action established by SF. It is synonymous with the materialization of a new intention in our mind. In other words, all the possible requirements of N^N materialize by the SF's complementary action, and an intention materializes in us by the Output Vector's action. Then, an additional requirement materializes in us, reflecting its intention, thereby an additional complementary action being established in SF. The structure to realize the above is the Synchronous Structure. The three-dimensional space of TDM represents our mind, whereas the three-dimensional coordinates represents N^N requirements. SF and we share the N^N requirements. In other words, the Input Vector commences to build the Synchronous Structure, and the Output Vector destroys it. Thus, the hypothesis is realized, which an intention materializes in us and then the N^N requirement reflecting it materialize.

The traditional program establishes logical action before executing on computer. In contrast, SF establishes it upon execution through the complementary action that is implemented autonomously by utilizing the state transition of Pallets, which appears structurally. As a result of execution, SF produces the same result as the traditional program, but it materializes a meaning fundamentally different from the meaning of movements of the traditional program.

7. Structure of tense control vector

In this section, the rules of each box of the predicate structure as the Signification Vector and the Action Vector are described. The relation of memory areas of the Signification Vector and the Action Vector is shown in Fig. 2.

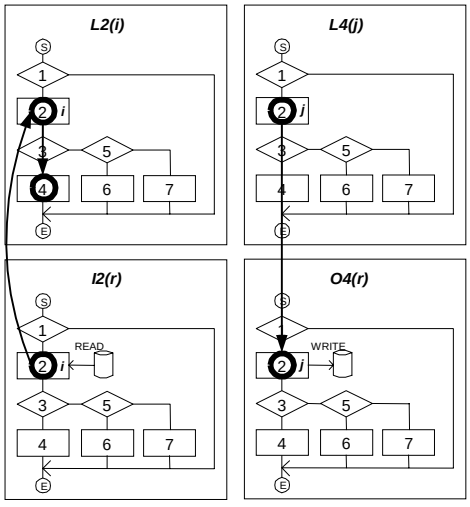

Figure 2. Relation of memory areas

7.1. Signification vector

The Signification Vector executes the self by itself by obtaining the execution-right from the Pallet Function. Upon the end of execution, the execution-right is returned to the Pallet Function. The Signification Vectors on *W04*, *W02* and *W03* have different roles as described below.

7.1.1. *W04* signification vector, *L4(j)*. The 1st rule compares and checks the presence of the data in the 4th area of itself and a value in the 4th area of *W03* Signification Vector of the same noun. If a self data exists in the 4th area of the *W04* Signification Vector, the purpose of this Signification Vector was accomplished, so execution-right is returned to the Pallet Function. A set of the 4th areas of *L4(j)* corresponds to the 2nd area of the Output Vector, *O4(r)*, which is the memory area for output. If there is no data in the 4th area of itself and if the data in the 4th area of W03 corresponds to the 1st rule of W04, the 2nd rule is executed. The 2nd rule is to operate an arithmetic calculation or duplicate the information existing in a specific memory area. This is the simplest requirement of users. In other words, the Signification Vectors can be defined by requirements on this level. The 2nd area of *L4(j)* is a temporary memory area of this Signification Vector. The 3rd rule checks the presence of self data in the 2nd area of *O4(r)*. If there is data in that place, the validity of the data shall be guaranteed by its presence. The reason is that the self in the 2nd area of *O4(r)* cannot materialize unless data of other nouns are present in the 4th areas of *L4(j)* or *L2(i)*, which are elements to materialize the complementary action to produce the data in the 2nd area of the *O4(r)*. The 4th rule of *L4(j)* duplicates this 2nd area data to its 4th area, which is the 2nd area of *O4(r)*, then the execution-right is returned to the Pallet Function. If there is no data in its 2nd area of *L4(j)*, the 5th rule checks the state transition of the Pallet, which is a set of status in the 4th areas on this Pallet. If there is a state transition, the 6th rule is executed. If there is no state transition, the 7th rule is executed. After either of the 6th rule or the 7th rule is executed, the execution-right is returned to the Pallet Function. The 6th rule sets a refusal flag which refuses re-execution of the Signification Vector into the its 6th area. It causes the issuance of a message if necessary. The 7th rule sets a recursion flag into its 7th area, which requests re-execution of the complementary action. The usage of this flag is not fixed in particular.

7.1.2. *W02* signification vector, *L2(i)*. The 1st rule checks the presence of data in its 4th area. If there is data, the purpose of this *L2(i)* has already accomplished, the execution-right is returned to the Pallet Function. If there is no data, the 2nd rule is executed. The 2nd rule sets data into its 2nd area after obtaining it from the input memory area, which is the 2nd area of the Input Vector. The 2nd area is a temporary memory area of this Signification Vector. The 3rd rule checks the presence of data in its 2nd area. If there is data, the 4th rule duplicates the data in the 2nd area into its 4th area, and the execution-right is returned to the Pallet Function. The 4th area of *L2(i)* is used by the complementary action of *L4(j)* and the 2nd rule of the *W03* Signification Vector. If there is no data, the 5th rule is executed. The 5th rule, the 6th rule and the 7th rule perform in the same way as those of the *W04* Signification Vector.

7.1.3. W03 signification vector, *L3(j)*. The 1st rule checks the presence of data in the 4th area of this *L3(j)*, and it makes judgment on the necessity of executing itself. If there is data in the 4th area, the purpose of this Signification Vector has already accomplished, the execution-right is returned to the Pallet Function. If there is no data in that place, its 2nd rule is executed. The 2nd rule is to specify necessary condition of the materialization of *L4(j)* of the same noun.

The requirement to determine this 2nd rule of SF requires logicality most strongly if the program is developed in the traditional ways. That is, this is the part requiring high-level work knowledge and a corresponding logical design. SF has its features at the point of simplifying this problem. That is, this logicality is resolved into the relation of data that establishes the way of mutual relation of plural nouns and their connection. In the traditional world, this must be replaced by a problem of establishing the whole by means of combining both of the sequential order that establishes parts of requirement and the sequential order of those parts.

In the scheme of SF, each noun is replaced Signification Vectors belonging to *W02* and *W04*, and data are determined for every noun through the complementary action which is implemented autonomously thereabouts. The complementary action is an action to manage the role for this sequential order. The meaning of its autonomous materialization is that the work to determine the sequential order can be removed from human work. The logical condition upon execution is built up resultantly by data for the respective nouns. Therefore, it is clear that this relation defined by the 2nd rule of *L3(j)* can be handled by the definition of a noun that affects the noun to become a variable. From the relation defined by the 2nd rules of *L4(j)* and *L2(i)*, a directory of nouns can be obtained. From the directory, nouns concerned with the 2nd rule of *L3(j)* can be obtained. If users define the way of

connection of the nouns as a requirement, instructions defined by this 2nd rule can be conclusively simplified much more than those defined logically by the traditional approach. In this connection, automatic algorithm is established, which converts traditional programs into SF. By using it, conversions from Assembler to COBOL and from COBOL to COBOL were implemented.

The 3rd rule checks the validity of data in the 2nd area, which is a temporary memory area for this Signification Vector. It is enough if the presence of data in the 2nd area is checked. If there is data in the 2nd area, the validity of data is guaranteed because of it. If valid, the 4th rule duplicates the 2nd area data into its 4th area, and the execution-right is returned to the Pallet Function. The 5th rule, the 6th rule and the 7th rules are treated in the same way as those of *L4(j)*.

7.2. Action vectors

The mode of giving and receiving of the execution-right is the same as the Signification Vectors. The rule of each box of the four kinds of Action Vectors is explained below.

7.2.1. Input vector, *I2(r).* The Input Vector can be defined if logical definitions of a set of nouns and DBMS are determined. The input memory area is placed in the 2nd area of *I2(r).* The 1st rule makes judgment if input data can be obtained into the 2nd area. That is, if the input memory area is in the initial state, and if a refusal flag of the *{I2(r)}* is off, data can be obtained, so the 2nd rule is executed. If not so, the execution-right is returned to the Pallet Function. The 2nd rule executes an input command such as READ. If this command is executed normally, it means that data has been obtained into the 2nd area. The 3rd rule makes judgment of the validity of the execution result of the input command. This means to examine the presence of data in the 2nd area, which is its input memory area. If there is data, it means that the execution of the input command has been implemented, the 4th rule sets a normal flag into the 4th area. After this, the execution-right is returned to the Pallet Function. If there is no data, the 5th rule is executed. The 5th rule, the 6th rule and the 7th rule are treated in the same way as that of *L4(j)*.

7.2.2. Output vector, *O4(r)* . The Output Vector can be defined if logical definitions of a set of nouns and DBMS are determined, same as the Input Vector. The output memory area is placed in the 2nd area. The 1st rule makes judgment if data in the 2nd area is to be output or not. That is, if a refusal flag of *{L4(j)}* is off, data in the output memory area, which is a set of the 4th areas of *L4(j)*, can be output, so the 2nd rule is executed. If not so, the execution-right is returned to the Pallet Function. The 2nd rule executes an output command such as WRITE. If this command is executed normally, it means that data has been output into the 2nd area. The 3rd rule makes judgment of the validity of the execution result of the output command. This is defined in accordance with the rule of, for example, DBMS. The 4th rule initializes its 2nd area and sets a normal flag in its 4th area if the result of executing the output command is valid. Then, the execution-right is returned to the Pallet Function. If the execution result is abnormal, the 5th rule is executed. The 5th rule, the 6th rule and the 7th rule are treated in the same way as that of *L4(j)*.

7.2.3. Structural vector, *S4(a).* The initialization of memory areas becomes indispensable so as to realize re-execution by the Pallet Function. The Structural Vector does area initialization required for it. However, the initialization of the *W04*'s output memory area is implemented by the Output Vector. A refusal flag and a recursion flag are not treated as an object of initialization. The initialization of these flags is implemented by the Pallet Function. The Structural Vector does initialization of other areas except above. If output success is declared by setting a normal flag in the 4th area of the Output Vector, it means that the output memory area has been initialized. Following the initialization of the 4th area of *O4(r)*, memory areas of *W03* and *W02* are initialized. Then, the Input Vector can start to operate.

7.2.4. Routing vectors, *R4, R2, R3R, R3D, R3M* and *R3C.* The Routing Vector specifies the next-to-execute Pallet in its 4th area. Which Pallet to specify is defined by the 2nd rule of the Routing Vector by using the state transition of the 4th area of *{L4(j)}* and a refusal flag of *{L4(j)}*. There are six kinds of Routing Vectors and each of which is explained below.

R4 is the Routing Vector of the *W04* Pallet. The 1st rule of *R4* checks the state transition of the *W04* to which the *R4* belongs. If there is no state transition, the *R4*'s 4th rule appoints *W02* of the same SF into the *R4*'s 4th area as a Pallet to be executed next, and the execution-right is returned to the Pallet Function. If there is any state transition, and the execution-right is returned to the Pallet Function. If the next-to-execute Pallet is appointed, the execution-right is returned to the Tense Control Function from the Pallet Function. If the next-to-execute Pallet is not appointed, the execution-right is transferred to all Tense Control Vectors belonging to the Pallet Function, and the Tense Control Vectors implement re-execution. If the re-execution of Tense Control Vectors continues, the *W04*'s state

transition will disappear. The transfer of the execution-right from *W04* to *W02* is a role of the Tense Control Function.

Vesting the execution-right in the Tense Control Vectors is a role of the Pallet Function, whereas returning the execution-right to the Pallet Function is a role of Tense Control Vectors. Vesting the execution-right in the Pallet Functions is a role of the Tense Control Function, whereas returning the execution-right to the Tense Control Function is a role of the Pallet Functions.

R2 is the Routing Vector of the *W02* Pallet. The 1st rule of *R2* checks the state transition of the *W02* to which the *R2* belongs. If there is no state transition, the *R2*'s 4th rule appoints *W03* of the same SF into the *R2*'s 4th area as a Pallet to be executed next, and the execution-right is returned to the Pallet Function. If there is any state transition, nothing is executed, and the execution-right is returned to the Pallet Function.

R3R, *R3D*, *R3M* and *R3C* are the Routing Vectors of *W03*. The 1st rule of *R3R* checks the state transition of the *W03* to which the *R3R* belongs. If there is any state transition, *W04* of the same SF is appointed into the *R3R'*s 4th area as the next-to-execute Pallet. If there is no state transition, and the execution-right is returned to the Pallet Function.

The 1st rule of *R3D* checks the state transition of the *W03* to which the *R3D* belongs. If there is no state transition, *W03* of the SF that is located just before this operating SF on PRD is appointed into the *R3D'*s 4th area as the Pallet to be executed next. If there is no state transition, the execution-right is returned to the Pallet Function. While there is state transition in *W03*, *R3R* handles the situation.

When termination of the operation of SF is not declared, the 1st rule of *R3C* checks the state transition of the *W03* to which the *R3C* belongs. If there is no state transition, *W04* of the succeeding SF on PRD is appointed into the *R3C'*s 4th area as the Pallet to be executed next. If there is state transition, nothing is executed and the execution-right is returned to the Pallet Function. While there is state transition in *W03*, *R3R* handles the situation.

When termination of the operation of SF is declared and if SF to be executed next is specified on PRD, *R3C* appoints W04 of the next-to-execute SF into the *R3C*'s 4th area. Thus, *R3C* are defined separately for the cases that termination of the operation of SF is declared and not declared.

If termination of the operation of SF is declared, *R3M* appoints into its own 4th area the *W04* of the preceding SF to return as the Pallet to be executed next. This preceding SF is located two or more SFs before this operating SF and is specified as the SF to be executed next on PRD.

8. Nature of noun

Meaning and functionality that cannot be realized with nouns extracted mechanically or with the complementary action can be realized by extending the concept of the noun. The extended concept of the nouns are called Boundary Noun, Equivalent Noun and Summation Noun. Propositions are made for each of these nouns.

8.1. Boundary noun

When three SFs, which are denoted as SF1, SF2 and SF3 respectively, make a row connected with *R3D* on PRD, SF1 and SF2 can be treated as one SF, and SF2 and SF3 can also be treated as one SF. However, it is uncertain that SF1 and SF3 possess a relation to materialize the Synchronous Structure. Therefore, if the 4th area of the Signification Vector of a certain noun of *W04* of SF3 is used for the complementary action of the Signification Vector of *W04* of SF1, there is a case in which the Synchronous Structure of SF1 does not materialize. In order to avoid this concern, the 4th area of the noun of the *W04* Signification Vector of SF3 is duplicated and placed also in the *W04* area of SF1. When the 4th area of a noun on W03 is justified, its Signification Vector sets the result not only into its 4th area but also into the 4th area of the same noun duplicated in the SF1. Such word is called Boundary Noun. Whether a Boundary Noun is required or not is determined by the 2nd rule of the Signification Vector of *W04* of SF1.

8.2. Equivalent noun

The purpose of *W04* Signification Vector is to determine output data. The way of the generation of the data happens to become plural depending upon the condition specified by *W03* Signification Vector. In this instance, one *W03* Signification Vector for the noun is defined, whereas *W04* Signification Vectors for the same noun for each way of the data generation should be defined separately. The noun having plural *W04* Signification Vectors is called Equivalent Noun.

8.3. Summation noun

There are nouns to be used for arithmetic summation. An auxiliary memory area for the exclusive use for summing up is added in the 2nd area of the *W04* Signification Vectors of those nouns.

9. Database

In the conventional ways of software development, the definition of data items of database is implemented logically, so there is a fundamental problem that logical adjustment between the database and the programs cannot be easily done.

SF is comprosed of Signification Vectors. Signification Vectors are determined for each noun. Signification Vectors are independent to each other. This gives universality to the logical relation between database and programs. Consequently, design work of database becomes easy.

10. Conclusion

The purpose of this paper is to present a summary of the scheme of TDM and the structure of SF to give a comprehensive view of the theory, and the some detailed information necessary for actual implementation is omitted for his purpose. The detailed information is shown at our web site.[2]

SF is created based on a non-engineering hypothesis. However, in the actual software development, high productivity and maintainability are witnessed. This is the first case that non-engineering hypothesis solves issues arising in the world of software engineering. The program structure of SF is completely different from that of conventional programs, which is an inevitable conclusion. The concept that the SF's structure shows is new and different from the engineering concept of software. For example, the pre-execution and pro-execution states are the same in the conventional program structure, whereas SF is different. The difference brings advantages to software developers. If the environments to determine SF, such as tools, OS and programming languages, are developed, the level of world of software would surpass the present one.

The hypothesis presented here is deeply influenced with the works of Spinoza, Leibniz and Wittgenstein but not based on engineering. This is why the author uses the term "non-engineering." This study is based on theory established in the Consciousness Model, and the theory will be presented at another opportunity.

REFERENCES

1. F. Negoro, "Principle of Lyee Software", in Proc. IS2000, Aizu, Japan, Nov. 2000, pp. 441-446.
2. F. Negoro, "A Proposal for Requirement Engineering", to be included in Proc. ADBIS'2001, Vilnius, Lithuania, Sep. 2001.
3. F. Negoro, "Intent Operationaliasation for Source Code Generation", to be included in the Proc. SCI2001, Orlando, USA, Jul. 2001.

[2] http://www.lyee.co.jp

Proceedings of the ACIS 2nd International Conference on Software Engineering, Artificial Intelligence, Networking & Parallel/Distributed Computing (SNPD'01). August 2001. Nagoya, Japan.