

Automatic Generation of Programs: An Overview of Lyee Methodology

Roberto Poli

Department of Sociology and Social Research, University of Trento

I-38100 Trento, Italy

and

Mitteuropa Foundation, I-39100 Bolzano, Italy

ABSTRACT

Lyee methodology for automatic generation of programs is discussed. Its conceptual background is presented and the platform's basic features are analyzed (predication, signification and action vectors, scenario function and process route vectors).

Keywords: software, automatic generation, object, process route vectors, Lyee

1. INTRODUCTION

Dreaming up applications is fun; writing down lines of code is often boring; debugging programs is thoroughly irksome. A number of recently developed software methodologies promise to keep us in the realm of dreams. Among others, the pattern movement and the extreme programming fraction deserve special mention (see [1] and [2]). Both are trying to speed up and simplify the process of software construction by changing its organization: the former by resorting to a few reusable conceptual schemas (not to be confused with the more formally-oriented idea of STLs); the latter by choosing to modify the working model of code writing (basically: everything should be written down by groups of two programmers working together). Two further, and more interesting, proposals have recently been put forward. Two research groups at least are dreaming up a platform able automatically to generate (almost) all the necessary code from pure data. Interestingly, the two proposals are substantially different from each other. The first, advanced by Kestrel Institute, a Californian-based research center, is a platform (named Specware) for the specification and formal development of software, and it is very heavily based on category theory. The second, advanced by Lyee Corporation, a Japanese-based group, is a business-oriented platform explicitly based on a few metaphysical principles (called the non-engineering hypothesis; see [7], [12]). The former is utterly mathematical; the latter is utterly philosophical. Both have found their way into serious applications (although Lyee seems to have been more widely tested than Specware). Both suffer from a serious lack of documentation, however (one may nevertheless consider [3] and [4]). This paper shall provide an introductory outline of Lyee.

2. BASIC DATA

Lyee, as previously said, is a software methodology explicitly

based on a series of metaphysical principles called the "non-engineering hypothesis." For those with an interest in theory, the presence of an explicit link between technological products and philosophical principles makes the platform very stimulating. Lyee's basic understanding is relatively straightforward, and can be summed up in the following theses:

1. Reality is a complex, continuous flux of ever changing forces and configurations. This basic situation is far too complex for our cognitive capacities to handle.
2. What we are able to do is filter this substrate space by crystallizing a few of its aspects. The everlasting complexity of the unknown substrate space can be managed by reducing its complexity ("density" in Lyee terminology).
3. We call the products of this intentional reduction of complexity 'objects' (Lyee terminology varyingly calls them 'intents', 'objects' or 'words').
4. Substrate space is never exhausted by our filtering of it. In other words, the substrate space can be indefinitely filtered. As soon as we modify the granularity of our filtering, a different set of objects is obtained.
5. Objects are static items. They compose the reduced-in-complexity space, and are the main building blocks of Lyee methodology.
6. Objects display a dual structure: they are a substance covered with qualities. Substance is something we cannot know analytically. Substance is what keeps together the set of qualities we call an object.
7. A three dimensional modeling space represents objects. The origin of the space represents the object's substance. The three dimensions represent (1) the input (what impacts on the object, the information coming from the outside environment to the object), (2) the object's internal structure (the way in which the object manipulates the information received), (3) the output (the object's answer, the information going from the object to the outside environment). In Lyee terminology, the three dimensions are respectively called W02, W03 and W04.
8. Objects perform actions and interact with each other. The series of the process route vectors establishes the object's behavior.

Given the above general points, the basic tenets of Lyee's methodology are the following: (1) collect your data and datatypes ('words' in their terminology), and (2) establish their connections and dependencies. The platform will take care of almost all the rest: it will produce the program, writing the

necessary code (up to 90% and in different languages, if required; [9]).

It is therefore apparent that Lyee is able drastically to reduce the figures in the software cycle: the available documentation claims that if Lyee is used, average development time can be reduced by up to one tenth, the need for documentation by up to one in a hundred, and maintenance by up to one in two hundreds. These claims alone - even if they are only partially borne out - will prove beyond doubt that Lyee deserves a very close look indeed.

The reasons for these outcomes are clear: since programs are automatically generated, their structure is crystal-clear and uniform. When something needs modifying, updating or revising, it is always apparent which part of the program should be considered. On the other hand, it is clear that programs generated in this way are substantially longer than traditionally written ones (usually 3-5 times longer; in the worst case known, even 50 times more). There may therefore be a problem of optimization.

Lyee methodology was invented by Fumio Negoro, and it has been successfully used to develop a number of heavy business applications. A new research center, supported by Catena Corporation, was opened a few years ago in Tokyo, and an international group of scholars, led by Issam Hamid, has recently been formed to explore the niceties and subtleties of Lyee.

Unfortunately, the available literature on Lyee is still rather cursory, incomplete and often poorly organized. It contains a substantial number of very awkward and unnecessarily original terminological choices. Moreover, no independent analysis has been conducted to date. This paper tries to remedy some of these shortcomings of the available literature by providing a basic readable introduction to Lyee's principles.

2. LYEE'S META-PRINCIPLES

Very often, contemporary software theory and practice seems to rely on *ad hoc* principles and constructions. Lyee tries to overcome this unfortunate state of affairs by looking at *what is universal* in software technology. This can be called Lyee's first meta-principle and it can be stated explicitly as follows:

Lyee's First Meta-Principle: Lyee methodology is based on what is universal in software construction.

Those with a grounding in contemporary mathematics will immediately recognize here the similar claim advanced by Bill Lawvere some decades ago: category theory, as a foundational theory, is based on "what is universal in mathematics" ([6], p. 281). Even if the mathematical bases of Lyee are seriously underdeveloped, the close parallel between their first meta-principle and the categorical viewpoint is nevertheless interesting (and may pave the way for comparison with the ideas developed by the above mentioned Specware platform).

Unfortunately, the mathematical claims advanced by some of Lyee's proponents (but not by Negoro himself) are often unconvincing, if not utterly wrong. This is a very delicate point that, if not dealt with, may severely detract from the appeal of Lyee.

On the other hand, by looking for what is universal, Lyee puts itself forward as a natural basis for many interesting developments. To mention but one, it may positively interact with the ontological movement so rapidly expanding in various fields of the information sciences (NLP and data-base theory among others; for an overview of contemporary works in ontology from an AI viewpoint, see [13]).

The second meta-principle adopted by Lyee is more complex in nature and requires a few sentences of explanation.

The basic ideas are that (1) the user alone knows what s/he intends and (2) his/her expressed intentions are nevertheless irreducibly vague and indeterminate. The latter is a structural condition and does not depend on the user's ability to express him/herself properly. In fact, ambiguity is a structural aspect of expressed intentionality.

According to Lyee, the only way to give precision to expressed intentions is to make them dynamic: run them and see what happens. Their dynamics give rise to products, and by connecting objects with the products of their dynamics the user can make his/her intentions more explicit and more precise. Ambiguities and other semantic difficulties are substantially reduced in this way.

In order to deal efficiently with the situation just described, Lyee distinguishes an *internal* side from an *external* side. The external side concerns objects (data, states, products, "words" in Lyee's terminology), that is to say static items. The internal side concerns processes (movements, acts, "tense control vectors" in Lyee's terminology), that is to say dynamic items.

The user should provide the basic objects to be handled, and their connections; Lyee itself directly governs all the rest (well, almost all of it).

The core of Lyee's dynamics is continuous iteration of its basic structure (to be explained below) until a situation of overall stability has been attained.

Activating instances, receiving a value, influencing and being influenced by other objects, changing: all these are processes. Lyee's internal algorithms govern the first three. The last is externally governed by modifying the original objects. When mainstream software languages are used, the latter may be a costly and difficult problem. But Lyee's internal structure makes it straightforward.

This situation is explicitly stated by the following meta-principle

Lyee's Second Meta-Principle: Ask the user for data and their connections only.

3. SOME TECHNICAL INFORMATION

Let's start According to Lyee methodology, the first step is to establish the set of data (objects or words) that an application should deal with. Next, programs for objects have to be established. Programs fix the objects' intended behavior.

The latter is governed by Tense Control Vectors, of which there are two different types: *Signification Vectors* and *Action Vectors*. The former concern simple objects; the latter concern complex objects (to be explained below). All the Tense Control Vectors share a common underlying structure called the *Predication Vector*.

Predication Vector Let us analyze the Predication Vector's structure by considering the easiest case, namely acceptance of an inputted value. The predication vector is the internal verification structure which establishes whether an input is to be accepted or rejected. As a matter of fact, the predication vector is barely more than an *if ... then, ... else* structure. Its fruitfulness depends on the architectural choice of assigning a *uniform control structure* to all the cases of object exemplification. This apparently trivial choice enormously simplifies the writing of the code.

Fig. 2 depicts the predication vector's structure when used as a signification vector. The meaning of the various boxes is apparent and does not require comment. The whole procedure in fact is rather straightforward. The expression "concretize the instance" has been used in order to avoid the cacophonous "instantiate the instance". For the program, concretized instances are real objects; they are things that it has to do with.

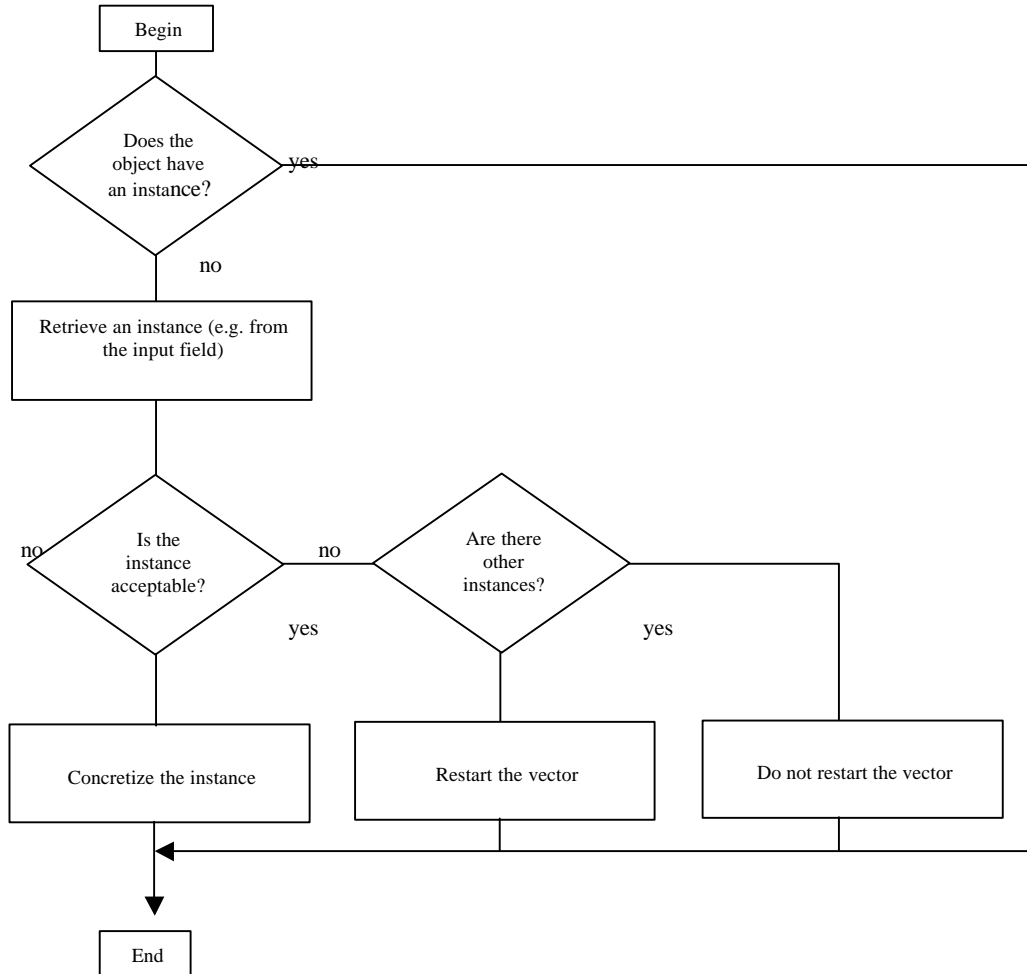
They have relations with other concretized instances, they influence them and are influenced by them. They exist (for the program). Moreover, they have a nature, they follow some pattern, namely the pattern of the object of which they are instances. The pattern's details are provided by the condition "Is the instance acceptable?"

Each object is defined by a fixed number of attributes. Moreover, Lye is very cautious in accepting data from the

outer world. It does not believe the content given until it has been checked against its instructions [15].

Objects are static items possessing specific granularities. Each predication vector instantiates an object (word or intent) [8]. Fig. 1 is a *prima facie* explanation of the Predicate Vector. Its role is nevertheless much deeper than this, for its main function is to allocate memory areas. An introductory paper like this may be permitted to leave this problem for another occasion.

Fig. 1



Signification Vectors Lye distinguishes three different signification vectors. The first case arises when the user inputs a value. At this point the predication vector is activated and a consistence check is performed. The signification vector becomes true only when the inputted value has been accepted (example examined in Fig. 1).

The other two cases deal with outputs and the manipulation of data. The main aspect here is that objects defined by signification vectors are independent of each other.

Action Vectors There are four different kinds of Action Vector:

- Input Vector (1)
- Output Vector (1)
- Structural Vector (1)
- Route Vectors (7)

Enclosed in parentheses is the number of their sub-kinds. Input and output are self-explanatory. The role of the Structural Vector will be explained soon. What really makes the difference are the Route Vectors (see below).

Action vectors are automatically generated as soon as objects, screens and their connections have been defined.

Input, Output and Structural Vectors Input vectors allocate memory areas and assign proper values to them. Output vectors copy the values of memory areas to external devices.

When an Output vector is activated, the Structural vector activates in its turn all the memory areas connected to it. Various subtleties are involved here which can be postponed to other occasions.

The Scenario Function and Its Pallets Collections of objects are called screens. Each object has its own signification vector. Their overall coherence is established and governed by the

Scenario Function. For convenience's sake, we may state the following correspondence

- object → signification vector
- screen → scenario function

The scenario function acts through pallets, distinguished as follows:

- W02: deals with the input
- W03: does various things, among them manipulating data (see below)
- W04: deals with the output

The whole basic series is therefore:

screen -(input)→ W02 → W03 → W04 -(output)→ screen

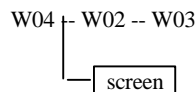
The transitions from one pallet to the next are governed by the Process Route Vectors. There are several of these, and I shall return to them below.

Before considering each pallet, one may ask why their numbering starts with #2. The reason is that pallet #1 deals with the so-called non-engineering hypothesis: that is to say, with Lyee's metaphysical assumptions. This is a serious reason indeed. Even so, I feel uncomfortable with the decision to start counting the technical components from #2. If the problem was how to distinguish the non-engineering hypothesis from the engineering proposal, why not label the former #0? In this case both engineering modules and the non-engineering hypothesis would receive a proper label, and the former could be correctly counted starting from #1.

A screen usually needs data from both pallets W02 and W04. In order to facilitate the drawing of diagrams, the basic structure is written in the following order:

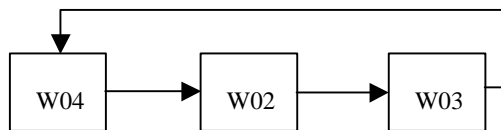
W04 -- W02 -- W03

This order enables the presence of a screen to be explicitly signalled:



Of the three, the most interesting pallet is W03, because real manipulation of data take place in its interior. Generally speaking, data from W02 are copied in W03, properly manipulated, and their result is copied in W04.

Process Route Vectors The basic dynamic structure is therefore the following



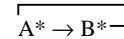
The basic structure is in a state of *recurrence* as long as W03 contains any state transitions. It only ends when there is no longer any state transition present in W03.

A basic structure #A may be linked to many other basic structures. The two obvious types of link are the serial and the parallel ones. An obvious example is provided by the situation in which a screen linked to a basic structure A may need

information from a database linked to a basic structure B. In this case, B can be seen as a sub-routine of A. To simplify the drawing of diagrams, basic structures be represented by A*, where * stands for its processes. A serial connection composed of three basic structures will therefore be represented in the following way: A* → B* → C*. Parallel connections are unproblematic as well.

Needless to say, the two kinds of connections can be linked together to form more complex structures.

It is also obvious that both require completion, as in the following diagram

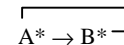


As already said, the basic transition rule is that a structure X iterates continuously until all the state transitions of its pallet W03 have been completed. As soon as W03 state transitions have been completed, the next structure B is activated, and so on.

Types of Process Route Vector The following list presents the various Process Route Vectors and briefly explains their meaning (Lyee's original terminology has been slightly modified).

- Route_42 Governs the W04 → W02 connection
- Route_23 Governs the W02 → W03 connection
- Route_34 Governs the W03 → W04 connection
- Route_30 Ends processing: W03 → End
- Duplex Governs the W03_i → W03_{i+1} connection
- Multiplex Governs the W03_i → W04_{i+1} connection.

To clarify the difference between Duplex and Multiplex links, let me repeat the last diagram of the previous section:



The indices present in the explanations of duplex and multiplex links can be read as providing a connection from basic unit B to basic unit A.

By this is meant that a duplex link from B to A starts from pallet W03 of unit B and ends in pallet W03 of unit A, whereas a multiplex link from B to A starts from pallet W03 of unit B and ends in pallet W04 of unit A. We may call these the intrinsic and the extrinsic connections, respectively. Intrinsic (duplex) connections are particularly important because they may reactivate W03's state transitions. In other and more traditional words, duplex links call subroutines.

Pallet Function The Pallet Function runs the pallet's Tense Control Vectors until all the pallet's objects are True and no True/False transition is scheduled. At this point, the Route Vector passes to the next pallet and the whole process starts again.

For the time being, the above information provides a reasonably *prima facie* exposition of Lyee. Needless to say, many details are still lacking and must wait for other occasions for explanation.

4. BASIC OBJECTS AND HIGHER-ORDER OBJECTS

Basic objects may form higher-order or complex objects. Like any other object, the latter may have their own form of unity, too: they act and react as synchronized wholes. In Lyee jargon,

basic objects are ‘words’ and higher-order objects are ‘logical units’.

Each higher-order object (logical unit) possesses its own scenario vector. The execution of the scenario vector synchronizes all its basic objects. We may therefore say that the basic objects instantiated by the scenario vector complete each other during any run of the scenario itself [9]. Moreover, each higher-order object possesses its own vector action. It acts as a whole through the actions of its parts.

Objects can only be synchronized through other objects.

Objects have two coordinates, according to Lyee. The first coordinate says whether the object is public (also named ‘regular’ and sigled T_0 , according to Lyee’s internal conventions) or private (also named ‘boundary’, and sigled T_1). Public means “visible to the user” (and therefore, on the screen, or in any other external device), whereas private means “internal to the program” (and therefore not visible to the user). The second coordinate is used for synchronization purposes. It duplicates a value from one object to another ([8], [9]).

Private or boundary objects are internal programs performing the work of local synchronization [10]. By local synchronization is meant the synchronization of the objects that should be synchronized according to the given task.

Lyee works by passing from local synchronizations to other local synchronizations, until everything has been synchronized. In other terms, no order of execution is wired into the program. Each piece of software is executed independently of any preestablished overall sequence. For this reason it is mandatory to run the system until all the inconsistencies have been removed, that is, until everything has been synchronized [15].

4. METHODOLOGY

[8] sums up the basic moves of Lyee’s methodology in a series of nine steps. The first one concerns the preparation of formularies, screens, and other public containers of information. In my wording, the various steps of the methodology should elaborate the following:

1. Formulary, screens, and other bearers (containers) of information (“definitive” in Lyee’s terminology);
2. Objects (“words”);
3. Process route diagrams;
4. Logical units;
5. Variables;
6. Vectors:
 - a. output,
 - b. input,
 - c. structural,
 - d. route;
7. Signification vector:
 - a. W04,
 - b. W02,
 - c. W03;
8. Tuning of the scenario function;
9. Simplification.

All the above have been explained except for items 5, 8 and 9. As to 5, I have been unable to grasp its intended meaning, whilst 8 is explained in none of the literature that I have been able to consult. On the other hand, the rationale of 9 is clear enough: since the programs automatically generated by Lyee are substantially longer than conventional ones, there may be various ways to reduce their size without overly scrambling their internal tidiness.

5. A FEW GENERALIZATIONS

According to Lyee methodology in its present form, the final user provides the objects and decides on their correct organization. This methodology has proved to be successful in various applications (business applications mainly), and has enabled substantial simplification of standard software methodologies.

The methodology may nevertheless run into serious trouble as soon as the range of possible applications is increased. Let us consider, among other things,

- IR applications, where the final user is typically different from the original author;
- Situations in which there are many users with potentially or actually conflicting requirements;
- Situations where the user him/herself does not know which decisions are best for his/her own tasks.

If Lyee aims to be a really general methodology for software construction, regardless of the specificity of any domain applications, its core should be enlarged so that the above cases (and many others as well) can be dealt with.

For this reason, the Lyee methodology in its present form may be considered a core methodology for default situations, those in which the final user is available and knows what s/he wants. If Lyee intends to become a thoroughly universal-purpose methodology, a series of new modules dealing with non default situations, like those listed above, should be added.

If we accept that intentions are somehow implicitly stored in the products of human activities and decisions, we should find a systematic way to make them explicit. Products in our sense can be written or spoken texts, artifacts, laws and norms, beliefs and attitudes.

A simple example may be helpful. Consider the fact that we all are usually able to partially or fully understand a text even when the author is neither available nor will be in any way available. Our capacity to (at least partially) understand his/her text means that at least some of his/her intentions have somehow been embedded in the text.

As is well known, *reverse engineering* tries to find out the technology embedded in a technological product. I am therefore proposing to develop Lyee by elaborating a kind of *reverse intentionality* to discover the intentions embedded in an intentional product.

Following this route, Lyee may encounter other methodologies, platforms or proposals that seek to explicate the *ontological structure* and / or the *cognitive structure* underlying the object under analysis [14]. A fruitful exchange may eventually rise.

6. CONCLUSION

Before concluding, I must confess that Lyee’s methodology needs clearer explanation. I have not found any step by step analysis of the basic moves of the methodology in their literature (a tutorial would be enormously helpful).

Moreover, the terminology is often abstruse and keeps shifting from one paper to the next. This must be remedied as soon as possible.

To my mind it is clear that Lyee are proposing some kind of intuition, but more explicit details will have to be provided before a scientifically acceptable proposal is forthcoming.

This said, I would add that Lyee has two great merits. The idea of giving all objects a basic universal structure (the predication vector), and the idea of distinguishing the basic processes (the process route vectors) as clearly as possible are major steps forward. Making these two features explicit has enormously simplified business-oriented software construction and maintenance. Whether this situation could be properly

generalized to other domains as well is still unclear. If it could, it would be fair to assume that some of the universal structures of software construction have been discovered.

7. ACKNOWLEDGMENT

The present paper was prepared as part of the Lyee International Collaborative Research Project, sponsored by Catena Corporation and The Institute of Computer Based Software Methodology and Technology.

8. REFERENCES

- [1] <http://www.cs.wustl.edu/~schmidt/patterns.htm>
- [2] <http://www.extremeprogramming.org/>
- [3] <http://www.kestrel.edu>
- [4] <http://www.lyee.co.jp>
- [5] I.A. Hamid and F. Negoro, "New Innovation on Software Implementation Methodology for the 21st Century", in *Proceedings of the 5th World Multiconference on Systemics, Cybernetics and Informatics (SCI2001)*, Orlando, USA, pp. 487-489.
- [6] W. Lawvere, "Adjointness in Foundations", *Dialectica*, 23, 1969, pp. 281-296.
- [7] F. Negoro, "Principles of Lyee software, IS2000, Aizu University, Aizu (Japan), pp. 441-446.
- [8] F. Negoro, "Intent Operationalisation for Source Code Generation", in *Proceedings of the 5th World Multiconference on Systemics, Cybernetics and Informatics (SCI2001)*, Orlando, USA, pp. 496-503.
- [9] F. Negoro, "A proposal for requirement engineering", in *Proceedings ADBIS2001, Advances in Database and Information Systems 2*, Vilnius, Lithuania, 2001.
- [10] F. Negoro, "The predicate structure to represent the intention for software", in *Proceedings of the 2nd conference on software engineering, Artificial intelligence, networking and Parallel/distributed computing (SNPD'01)*, Nagoya, Japan.
- [11] F. Negoro, "Methodology to Define Software in a Deterministic Manner", forthcoming.
- [12] F. Negoro and Issam A. Hamid, "A proposal for intention Engineering", *Proceedings SSGRR2001*, Rome.
- [13] R. Poli, *ALWIS. Ontology for Knowledge Engineers*, Utrecht, 2001.
- [14] R. Poli, "Perspective shifters", forthcoming.
- [15] H. Ryosuke, T. Yozo, and T. Shigeaki, "Software as Data Model", forthcoming.