Requirements Modeling in Agile Methodologies with X-Machines

A. LIPSKI
lipski.artur@gmail.com

Faculty of Cybernetics, Military University of Technology
Kaliskiego Str. 2, 00-908 Warsaw, Poland

The demand for more complex but also more reliable and correct computer based systems on the one hand, and the fact that several changes in the user requirements through the development cycle on the other hand, leads to the need for more formal but also agile development methodologies. X-Machines is an intuitive formal method which can be easily applied to agile methodologies, especially in specification phase, gaining a lot of advantages.

Keywords: X-Machine, agile methods, requirements modeling.

1. Introduction

It is proved that traditional approach of software development process fails to cope with even small changes in requirements at any stage after the analysis phase. It is likely that in large industries the most popular software development methodology is the waterfall process, which however exhibits an awkward behavior in changes during the later stages of the development. This is mostly the reason for the dramatic increase in time and cost of the development of the software.

To develop reliable, high integrity and correct systems it is said that companies which produce software should use formal methods to achieve this. But, it is also a common belief of people outside the formal methods community that formal methods are difficult to understand and to use. Furthermore formal specification can be costly and time consuming. Also formal methods do not cope well with late changes in requirements which could result in major rework of the produced software.

In order to be useful to modeling of computer base systems a formal method should be able:

- to model both the data and flow of the system
- to introduce a practical, modular, disciplined way of modeling that will facilitate the modeling of large scale systems
- to be intuitive, practical and effective towards implementation of the system
- to facilitate development of correct system.

One of formal methods is common to fulfill all above aspects. This formal method is so called X-Machine introduced by Eilenberg [1] in 1974 which has been shown to have the computational power of a Touring machine. In 1988 Holcombe [4] proposed X-Machines as a basis for possible specification language which is capable of modeling both the data and flow of the system. With the development of a formal testing strategy, a formal verification technique and a methodology of building communicating systems out X-Machine components, and with an added support of tools and the proposal of a formal framework for the development of more reliable systems, most of the above mentioned requirements are met with emphasis in the development of correct systems.

Over the last years agile methodologies have been introduced which attempt a useful compromise between no process and too much process, providing just enough to gain a reasonable effect. Some of the proposed agile methodologies are: Extreme Programming (XP), Feature Driven Development (FDD), SCRUM, Dynamic System Development Method (DSDM), Agile Modeling (AM), etc.

This article will show what X-Machines are and how to use them with one of the popular agile methodologies to model requirements in the way to achieve correct system.

2. X-Machines

X-Machine is a formal method which provides a diagrammatic approach of modeling the flow of the system by extending the expressive power of Finite State Machine (FSM). Transitions between states are not expressed just by a simple input symbols but through a set of functions. X-Machines are capable of modeling non-trivial data structures by using a so called memory, which is attached to the X-Machine. Functions receive input symbols and memory values, and produce output while modifying a memory values.

The X-Machine model has many advantages [8]:

- it is formal
- it is rigorous
- it is expressive
- it provides unambiguous models
- it is capable to capture both static and dynamic system information
- it is based on a fully general and formalized computational model, that could form the basis of an universal approach to design of systems
- it can support component based development
- it is supported by appropriate tools.

2.1. Definition of X-Machines

As it was said earlier X-Machine is essentially a finite state machine whose label are elements of R(X), where R(X) is a relational monoid on X. A particular class of X-Machine is a stream X-Machine which is defined as follows: For any set A, A^ε denotes the set $A \cup \{\varepsilon\}$, where ε is an empty sequence. A* denotes the free monoid generated by A [5].

Definition 1. *A stream X-Machine is a tuple*

$$X = (\Sigma, \Gamma, Q, M, \Phi, F, I, T, m^0) \qquad (1)$$

where:

- Σ and Γ are finite sets called the input and output alphabets respectively
- Q is the finite set of states
- M is a (possibly infinite) set called memory
- Φ is a finite set of partial functions of the form: $f: M \times \Sigma \to \Gamma \times M$
- F is the next state partial function $F: Q \times \Phi \to 2^Q$
- I and T are the sets of initial and final states
- m^0 is the initial memory value.

We define a *configuration* of the X-Machine by (m, q, s, g), where $m \in M, q \in Q, s \in \Sigma^*, g \in \Gamma^*$. A machine computation starts from an initial configuration, having the form $(m^0, q^0, s^0, \varepsilon)$, where $q^0 \in I$ is an initial state and $s^0 \in \Sigma^*$ is the input sequence. Figure 1 shows a model for abstract X-Machine.

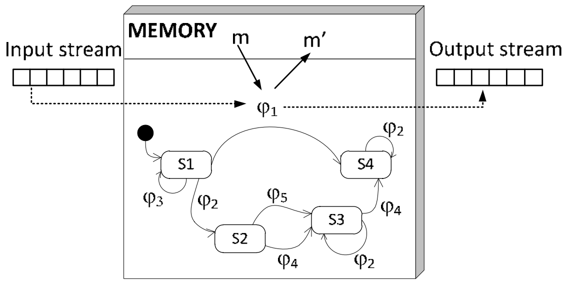

Fig. 1. Abstract X-Machine model [5]

Definition 2. *The output corresponding to an input sequence. A change of configuration, denoted by* $\vdash$*:*

$$(m, q, s, g) \vdash (m', q', s', g') \qquad (2)$$

is possible if:

- $s = \sigma s', \sigma \in \Sigma$
- there is a function $f \in \Phi$ emerging from *q* and reaching q', $q' \in F(q, f)$, so that $f(m, \sigma) = (\gamma, m')$ and $g' = g\gamma, \gamma \in \Gamma$

$\vdash^*$ *denotes the reflexive and transitive closure of* $\vdash$.

For any $s \in \Sigma^*$, *the output corresponding to this input sequence, computed by the stream X-Machine X is defined as:*

$$X(s) = \{g \in \Gamma^* \mid \exists m \in M, q^0 \in I, q \in T, \text{ so that } (m^0, q^0, s, \epsilon) \vdash^* (m, q, \epsilon, g)\} \qquad (3)$$

Definition 3. *A stream X-Machine is called deterministic if:*

- I contains only one element: $I = \{q^0\}$
- $F: Q \times \Phi \to Q$
- if f, f' are distinct arcs emerging from the same state, then $\mathrm{dom} f \cap \mathrm{dom} f' = \emptyset$.

Definition 4. *A stream X-Machine is called minimal, when its associated automation* $A = (Q, \Phi, F, I, T)$ *is minimal.*

Definitions quoted above are the main definitions in X-Machine theory and are true for all kinds of X-Machines. When there is a need to model complex systems with dynamic interactions between its components, usage has so called Communicating X-Machines System.

Definition 5. *A Communication X-Machines System (shortly CXMS) with n components is a 4-tuple* $CXMS_n = ((P)_{i=1,\dots,n}, C, C^0, O)$, *where:*

- P_i is the X-Machine with number i, $P_i = (\Sigma_i, \Gamma_i, Q_i, M_i, in_i, out_i, F_i, I_i, T_i, m_i^0)$
- C is a matrix of order $n \times n$, used for communication between the X-Machines
- C^0 is the initial content of C
- O is the output tape of the system; at any time, the content of O has the form

$(g_1, \ldots, g_n) \in \Gamma_1^* \times \ldots \Gamma_n^*$. For any i, O_i will denote the output tape of the i^{th} component of the system.

Definition 6. *A communication function is a function:*

$$cf(s, m, in, out, c) = (s, m, in', out', c') \quad (4)$$

where:

$$m \in M, in, in' \in IN,$$
$$out, out' \in OUT, c, c' \in C.$$

Definition 7. *A processing function is a function:*

$$pf(\sigma, m, in, out) = (\gamma, m', in', out') \quad (5)$$

where:

$$\sigma \in \Sigma, m, m' \in M, in, in' \in IN,$$
$$out, out' \in OUT, \gamma \in \Gamma.$$

Fig. 2 shows abstract model of a Communication X-Machine.

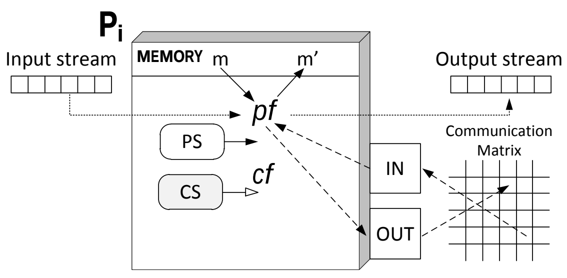

Fig. 2. Abstract example of Communication X-Machine model[5]

2.2. Extreme X-Machines

There is also a special kind of X-Machines called Extreme X-Machines (XXM) [9], which has been design to be both simple and flexible to use in agile scenarios.

An XXM is an extended form of the X-Machine which allows each function to be guarded by a predicate which identifies a state of memory, with the caveat that from the origin state there must also be a transition defined for the inverse predicate. The XXM has complex memory structure, allowing to specify variable names, arrays and arrays within arrays. The XXM within states are limited such that they run when the state is entered and when they exit the state is exited.

By using a predicate as a guard function on a state transition it is possible to separate the memory that is needed from the rest of the memory. In case where this memory state defines a database, it means that there is no need to consider many possible states of the database and instead define what a valid state might look like.

Extreme X-Machines are mostly used in requirements engineering phase to represent elements of the user interface which helps in separating the user interface code form that of system control code.

3. From requirements to X-Machines

Software requirements can be captured from users and documented in many ways. Each agile methodology has its own technic to document requirements. For example Unified Process (UP) uses Use Cases and Extreme Programming uses Story Cards and Story Boards [2], [3].

No matter which agile methodology and requirements capture technic is used, to transform a requirement to the corresponding X-Machine, firstly the states and the transitions, then memory structure, the input and output sets, and finally the transitions functions should be defined.

3.1. States and Transitions

States and transitions are derived by examining the steps both the main success scenario and the extensions:

- there is always an initial state corresponding to that state in which requirement scenario makes its start. In use case driven development it would be a moment in which the actor triggers the use case
- the final state, which usually coincidence with the initial state, is notated as a different node in the state machine to emphasize the end or termination of requirement scenario
- each user interaction introduces a new transition leading to a new state in the X-Machine state diagram
- each extension introduces a new transition from the same starting state of the previous user interaction. The transition leads to a new state, if an interaction follows in the extension steps. If there is no user interaction in the extensions, unless otherwise stated, the transition loops back to the same state.

Each user interaction and all its subsequent system functions until the next user interaction are modeled by a single processing function. So a processing function does not only model the user interaction but also all the processing that guarantees that all subsequent system functions are correctly executed. The processing function is not always triggered by the interaction but it contains, as guard expression, all the conditions that have to be satisfied for the transition to occur.

3.2. Memory

The memory structure and contents cannot be directly derived from the requirement text. The object-flavored memory structure is easily derived from the domain model which is usually represented as a static class diagram. A domain model presents the concepts of the problem domain and their relationships. Concepts are represented as classes with attributes and relationships as associations between classes.

Since memory has to be filled with specific values, we define sample objects for operating the machine. A small number of objects is used in order to exercise different scenarios.

3.3. Input and Output Sets

In use case driven development inputs can be directly derived from the system sequence diagram. In turn output set is usually defined as a set of messages, that are displayed after each transition.

3.4. Processing Functions

For each processing function there should be defined the input and the memory state that trigger the function, the output that the function produces and the memory update.

4. A X-Machine example

Let's say that there is a need to model a behavior of a vending machine. The usual procedure of a vending machine is that customer enters some coins (10gr, 20gr, 50gr, 1zł, 2zł, 5zł), selects a soft-drink (coke, sprite, fanta, nestea) and finally presses a button to execute the order (enter button). The prices of the drinks are stored in some database that is accessed by a set of external data processing functions related to the model.

Figure 3 shows below a X-Machine diagram for vending machine from example.

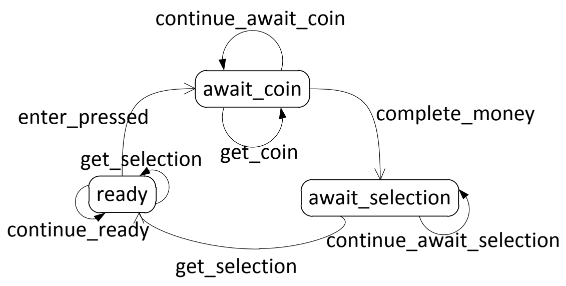

Fig. 3. X-Machine state diagram for vending machine

4.1. Memory example

A partial class diagram for the example of X-Machine model of vending machine is shown in Figure 4.

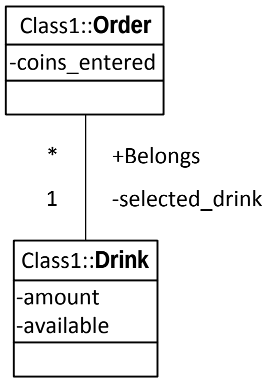

Fig. 4. Partial class diagram of vending machine

For each X-Machine we have to define the initial memory state. The memory instance below could be the initial memory state of X-Machine.

M={Order, Drink}
Order={{order1, order2}, {coins_entered, selected_drink}}
Drink={{coke, sprite, fanta, nestea}, {amount,available}}
where
coins_entered={10, 20, 50, 100, 200, 500}
amount={150,200}
available={drink, no_drink}
order1.coins_entered=0,
order1.selected_drink=null,
order2.coins_entered=0,
order2.selected_drink=null,
coke.amount=150, coke.available=drink,
sprite.amount=150, sprite.available=drink,
fanta.amount=150, fanta.available=no_drink.
nestea.amount=200, nestea.available=drink

4.2. Input and Output Sets example

In example machine's input refer either to the coins (CoinType), or to the drink selected (SelectionType), or finally to a button (ProcessType) which executes the order.

Input={CoinType, SelectionType, ProcessType}
where
CoinType={10, 20, 50, 100, 200, 500}
SelectionType={coke, sprite, fanta, nestea}
ProcessType={enter}

The machine's output consists of three basic elements:

- the current state of the machine (State)
- the amount of money inserted so far (AmountType)
- a message (AvailableDrinkType) informing the user whether a drink is available or not.

Output=
{State, AmountType, AvailableDrinkType}
where
AmountType={150,200}
AvailableDrinkType={drink, no_drink}
State={await_coin, ready, await_selection}

4.3. Processing Functions example

Below there is a definition of a processing functions of the vending machine.

get_coin(x, (amount, selection)) =
if (x $\in$ coin_type and $\neg$sufficient(x, amount)) then ((await_coin, add(x, amount), no_drink), (add(x, amount), selection))

5. Advantages of using X-Machines

Using X-Machines in combination with agile methodologies to model system requirements has several valuable advantages:

- the X-Machine method supports a disciplined modular development, allowing the developers to decompose the system under development. Decomposition aids a value in handling large scale system modeling
- X-Machines provide the appropriate style of developing models reusing off-the-shelf existing verified component models that where made earlier
- formal modeling provides that description are unambiguous. The developers understand system better as a result of trying to describe it unambiguously. An intuitive formal method, which X-Machine is, makes possible for the simple user to understand the documents produced in modeling phase. This enhances the communication between the user and the development team allowing feedback from the users
- tools, like automatic animation of the model, help user to monitor the model proposed by the development team allowing more complete and immediate feedback. As a consequence at the end of each iteration the user provides valuable feedback early in the development process
- the development team and the users learn while developing with agile methodologies, and all learn fast due to the intuitiveness of the X-Machine formalism, which improves the whole process
- changing requirements is (must be) an excepted event through the development time and can be handled efficiently by the modularity of the X-Machine model (especially in communicating X-Machine model) and the flexibility of the X-Machine component model.

6. Conclusion

The aim of this article was to explain what are X-Machines and how to model software requirements for agile methodologies with them.

X-Machines provide a powerful and easy in use formal tool for capturing and documenting requirements. Different types of X-Machines make it easy to model software requirements from user interface behavior (Extreme X-Machines), ends on internal software components and dynamic connections between them(Communicating X-Machine model).

To formalize agile requirements using X-Machines, the structure of the memory was derived from the domain model. By doing so, the domain model was also validated, since an incomplete domain model would not allow the execution of X-Machines. Thus formalization also bridges the functional world of requirements with the object-oriented world of class diagrams used in the domain model.

7. Bibliography

[1] S. Eilenberg, *Automata, languages and machines*, Academic Press, New York, 1994.

[2] K. Beck, *Extreme programming explained: embramce change,* Addison-Wesley, 2000.

[3] A. Cockburn, *Writing Effective Use Cases,* Addison-Wesley, 2000.

[4] M. Holcombe, *X-Machines as a basis for dynamic system specification*, Software Engineering Journal, 1988.

[5] M. Holcombe, F. Ipate, *Correct Systems: Building a Business Process Solution,* Springer Verlag, Berlin, 1998.

[6] M. Stannett, *The Theory of X-Machines – Part1*, University of Sheffield Regent Court, United Kingdom, 2006.

[7] M. Holcombe, ”An integrated methodology for the formal specification, verification and testing systems”*, Proc. EuroSTAR 93*, London, 1993.

[8] H. Gheorgescu, C. Vertan, ”A New Approach to Communicating X-machines Systems”*, Journal of Universal Computer Science*, Vol. 6, No. 5, 2000.

[9] C. Thomson, W. Holcombe, ”Applying XP Ideas Formally: The Story Card and Extreme XMachines”, *Proceedings of 1st South-East European Workshop on Formal Methods*, Thessaloniki, Greece, 57−71, South−East European Research Centre, 2003.

Modelowanie wymagań w metodach Agile z wykorzystaniem X-Machines

A. LIPSKI

Popyt na bardziej złożone, ale również bardziej wiarygodne i prawidłowe systemy z jednej strony, oraz fakt, że klika zmian w wymaganiach użytkownika w trakcie cyklu rozwoju oprogramowania z drugiej strony, prowadzi do konieczności użycia bardziej formalnych ale również zwinnych metody wytwarzania oprogramowania. X-Machines to intuicyjne formalne metody, które można łatwo zastosować wraz z metodami Agile, zwłaszcza w fazie specyfikacji wymagań, osiągając wiele zalet.

Słowa kluczowe: X-Machine, metody Agile, modelowanie wymagań.