DESIGN OF A MODELLING LANGUAGE FOR SEMIOTICS BASED AGENT SYSTEMS

Mangtang Chan
Department of Compute Science, City University of Hong Kong, 83 Tat Chee Avenue, Hong Kong SAR, PRC

Keywords: Semiotic agents, modelling language, semiotic framework, model-driven development.

Abstract: Semiotics has been used to describe computer programming and systems since 1960. The use of semiotics in information system development has not yet been seen as a mainstream paradigm although a methodology MEASUR has been established based on the semiotic framework. To evangelise the use of the semiotic framework, this paper defines a modelling language which supports systems specification by applying the Semantic Analysis and Norm Analysis methods in MEASUR. During the design of the language, the Model-Drive Development approach is used as a reference and a meta-model of semiotic agents is defined. The detail language constructs are presented and illustrated with an example of inter-agent tracking.

1 INTRODUCTION

Discussion of relating semiotics to computers can be dated back to as early as 1960's when Zemanek (1966) examines programming languages from the viewpoint of semiotics. The three fields or dimensions (the term used by Zemanek) of semiotics: pragmatics, semantics and syntactics are used to understand programming languages. A programming language always has an interpreter, could be human or computer that would execute the program. The interpretation and whatever result following would be the pragmatics, there is always semantics about the signification of the program text unless the programming language is not a meaningful one, and syntactics would be the way of how symbols or characters are combined to form the language.

Andersen also presents semiotics as the framework for understanding and designing computer systems as sign systems, as targets of interpretation (Andersen 1991). According to him, in the total picture of a computer system, semiotic activities could be found from the top down to the very bottom of the system. A system could be specified by a program text, a sign to the compiler or interpreter which is also signs themselves. Program execution is a process of interpretation of the machine code by the computer processor. To the programmer, the program text on one hand would be transformed to assembly code and then to machine code, on the other hand the program could also be interpreted and new software concepts would be created, such as statements, variables, lists, loops, objects and modules. Repeated creation and interpretation of programs would form a computer system. This semiotic perspective from programs to computer systems provided a logical extension of the description proposed by Zemanek.

Stamper (2000) suggests a "new" direction for systems analysis and design, and argues that existing information systems analysis and design methodologies since 1950 have been based on an information flow paradigm which is data centric and makes people think with a technical bias. He proposes an information field paradigm - "An information field is established by a group of people sharing a set of norms". Norms are units of knowledge expressed as rules based on which subjects to whom the norms applied would act when a certain sign occurred. He states that information systems are organized behaviour with constant interplay between signs (information) and norms (knowledge). This paradigm leads to the theory of information systems as social systems which is able to underpin different kinds of systems with or without the use of computers. Liu applies this paradigm to information system engineering based on the semiotic framework and the use of MEASUR,

Chan M.
DESIGN OF A MODELLING LANGUAGE FOR SEMIOTICS BASED AGENT SYSTEMS.
DOI: 10.5220/0003267200950102
In *Proceedings of the Twelfth International Conference on Informatics and Semiotics in Organisations (ICISO 2010)*, page
ISBN: 978-989-8425-26-3

a set of norm-oriented methods for business systems modelling and requirement specification for information systems (Liu 2000; Stamper 1994).

The semiotic framework has not yet been seen as a mainstream methodology in the software industry. Barry & Lang (2001) conducts a survey covering multimedia and Web development techniques, and methodologies used in companies involved in large-scale, in-house, data-heavy business applications. The survey concludes that practitioners are not using models cited in the literature or research work. This could probably be explained by the fact that use of the models demands in-depth understanding of relevant knowledge and the lack of tools to map the theoretical model to implementation. On the other hand, commercial programming languages are only applicable to a specific problem domain or technology platform with limited portability and they usually work at a low level of abstraction. This paper aims at solving this dilemma by finding a modelling technique that is based on a sound theoretical framework but at the same time can be easily understood and translated into proper implementation of different types of real-life applications in different problem domains as well as technology platforms.

In the past few years, the Model Driven Development (MDD) has gained substantial attention among practitioners with the Unified Modelling Language (UML) (OMG. 2009) from OMG and the Specification and Description Language (SDL) (ITU 2000) from ITU as two prominent examples. A model is defined as a collection of artefacts that describes the system (Balmelli, et al. 2006). An artefact is defined as any item that describes the architecture, ranging from diagram, document or specific language designed for the description. MDD uses the technique of abstraction to handle complex system modelling. A system model can be looked at from different viewpoints and abstraction levels. An abstraction or model level is therefore a subset of the overall model that presents a particular focus of the system. Specification of a system with a design language is a representation in the form of a model consisted of different language constructs. The language constructs are based on lower level models (meta-models) and the specification has to be translated into implementation which is also another kind of model. This series of model transformations is the essence of model-driven development. Our work is an attempt to evangelise the use of semiotics in information systems development and designing a modelling language is a first step. The MDD approach is also adopted because it is well understood in the industry.

2 SEMIOTIC FRAMEWORK

2.1 MEASUR

The modelling language is based on the semiotic framework and MEASUR. MEASUR as a methodology provides five methods for information systems development:

- Problem Articulation Methods (PAM) - to be applied at the early stage of a project with the aims to identify related agents and an action course; to reveal the cultural behaviour; to structure the problem into a main course action and its surrounding collateral activities and to identify norms that govern agent's behaviour in the system;
- Semantic Analysis Methods (SAM) - this is basically the ontological dependency analysis to explicate in a precise form the relationship between words and appropriate actions used in the system;
- Norm Analysis Methods (NAM) - this is to specify the patterns of behaviour of the agents in the form of norms in which responsibilities of agents are defined, conditions in which some actions can or cannot be performed by agents;
- Communication and Control Analysis - this is to analyse communications between agents identified by PAM through a classification of messages into groups of informative, coordinative and control according to the intention of sender agent;
- Meta-Systems Analysis - this is to deal with the meta-aspects of a project such as planning and management.

Our work focuses on the modelling part and is therefore based on the SAM and NAM.

2.2 Semiotics and the Information Systems Development Cycle

Our aim is to use a semiotics based modelling language to build information systems instead of only perform the analysis and design. Liu describes approaches of combining semiotics with other system analysis and design techniques in going through different stages of information systems development with different activities. Table 1, modified from Liu's work (Liu 2000), summarises

Table 1: Conceptual viewpoints in adopting semiotics for different stakeholders and IS activities.

IS activities	Personnel involved	Option 1	Option 2	Option 3
		Conceptual Viewpoints		
Requirement analysis	Users, analysts	Agents, Affordances, Norms	Agents, Affordances, Norms	Agents, Affordances, Norms
Systems analysis	Analysts	Agents, Affordances, Norms	Data flow diagram (if structured analysis methodology is used)	Agents, Affordances, Norms
Systems design	Analysts, developers	Agents, Affordances, Norms	Entity-Relationships	Agents, Affordances, Norms
Systems implementation	Developers	Normbase	Relational database, programming languages, CASE tools	Agents, Affordances, Norms
Systems execution	Users, maintenance programmers	Normbase (as e.g., relational database)	Programs, database or files	Agents, Affordances, Norms

the possibilities. The activities are requirement analysis, system analysis, design, implementation and systems execution. The conceptual viewpoints used by different activities are compared for three options: Option 1 described in the original work, Option 2 to show the extreme case where totally different methodologies and hence viewpoints are used in different activities, and Option 3 which is the option taken in our work. It could be noted that Option 1 uses semiotics as the theoretical view point throughout the activities until systems implementation when Normbase is used. Normbase is a software environment for managing norms and agent semantics resulted from SAM. It is interesting to note different types of stakeholders would use different conceptual views in different activities in Option 1 and 2. Because of the change in conceptual view point, the implemented systems would not be seen as agents functioning with the semiotic principles. Our approach retains the use of the semiotics view point in all activities and for all stakeholders up to the point of systems execution.

We argue that this would offer advantages to all stakeholders with the main one being that they can understand each other by using a common model. The result of this approach is that the software used in systems execution are actually software agents governed by norms.

2.3 Semiotic Agents

The term semiotic agent is not explicitly used in the work of Stamper and Liu but has been found used by Joslyn to model socio-technical organizations which are defined as large number of groups of people hyperlinked by information channels and interacting with computer systems which in turn interacted with a variety of physical systems (Joslyn & Rocha 2000). Semiotic agent is proposed by Joslyn as an agent-based modelling technique to model and simulate emergent decision structures in command and control organization (e.g. 911/emergency response systems). Design of these semiotic agents is based on the reference and interpretation of sign tokens. Characteristics of semiotic agent, according to Joslyn, are:

- Capable of measuring a certain of part of the environment and this perceived part of the "world" gave the repertoire of behaviour and hence its field of knowledge;
- Capable of evaluation of current status to judge based on its own belief and determine its own action;

- Able to access to a stable, decoupled memory with which interactions with other agents can be stored; the agent would use this memory in its evaluation of action behaviour;
- Asynchronous in behaviour and would not perform actions in constant time-steps with other agents; it would respond to discrete-event clues or follow a schedule of sequential interactions;
- Able to communicate by the creation, transmission, receiving, storage and interpretation of tokens based on the existence of environmental tokens and regularities which follow the laws of the environment and agent rules;
- Able to share certain amount of knowledge instead of relying on solely on individual rules or knowledge bases.

Agents in the semiotic framework, although based on organizational semiotics are essentially having the same characteristics. We adopt the term semiotic agents to refer to the software agents which are the main building blocks in our model and systems specified by the modelling language are therefore multi-agent systems.

3 DEFINING THE MODELLING LANGUAGE

3.1 The Meta Model

We name the modelling language SAME-ML (Semiotic Agent Modelling Environment Mark-up Language). Systems modelled by SAME-ML consist of one or more environments which contain one or more semiotic agents. The semiotic agents are capable of concurrent execution. Each agent would continuously look for signs in the environment from its own perspective. The norms governing the behaviour would be implemented by rule sets which could be interpreted and checked with a rule engine. Therefore the structure of a semiotic agent is defined as a set of affordances implemented as functions, methods or services depending on the implementing technology; a set of signs, a set of rules and a rule engine. The agent would continuously update the sign set and assert it as facts (rule based system terminology) to match against the rule set, the processing is done by the rule engine which would fire actions when certain rules are matched. Actions are affordances of the agent. Figure 1 depicts the abstract structure of systems produced by SAME-ML.

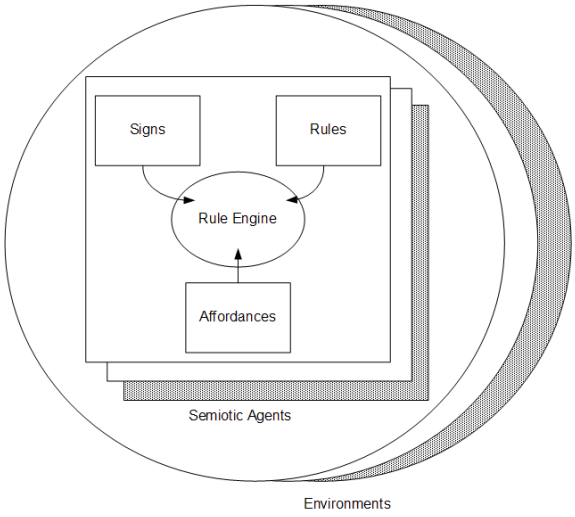

Figure 1: Abstract structure of Semiotic Agents.

3.2 XML and XML Schema

We use Extensible Mark-up Language (XML) to define SAME-ML. XML is a standard proposed by W3C (World Wide Web Consortium) for describing the structure of documents. The structural information is marked by elements in the form of tags with the format <tag> ... </tag>, known as the open and close tags. Data are put inside the tags. Properties known as attributes can also be specified for the tag, such as <tag attribute="value">. Elements can further be included within an element to form any hierarchical structure. By looking at the enclosing tags and according to their defined meaning, the data can be categorized and processed. The main difference between XML and other mark-up languages (e.g. the HTML - Hyper Text Mark-up Language used in Web pages) is that the meaning of the tags is extensible (Zisman 2000). The interpretation of the tags is not fixed but based on another document that defines the meaning of tags. By changing the document, different mark-up languages can be defined for different application domains. The definition document itself has to be written in another language which serves as a data definition language, schema language or meta-language in the process of defining the mark-up language. This definition document is in fact a transcription of the meta-model for the language. We choose XML Schema to specify the meta-model of SAME-ML. XML Schema, also known as XSD (XML Schema Definition), itself is an XML. The whole specification of the meta-model is fairly complex and we only use example SAME-ML to

show the language structure in the following sections. More detail information about XML Schema can be found in relevant literatures (P. V. Biron &A. Malhotra. 2009; H. S. Thompson, D. Beech & M. Maloney. 2009).

3.3 Detail Language Structure

3.3.1 The System

In classical semiotics, agents coexist together in an organization interacting with each other and with the environment. The organization can be regarded as an information system. SAME-ML uses the highest level element <system> to enclose all other sub-elements. The system and the agents must be named and the system must consist of at least one or more agents.

```
<system name="  ">
  <agent name="  "> ..... </agent>
  <agent name="  "> ..... </agent>
  ......
</system>
```

3.3.2 Agents

In our meta-model, semiotic agents have one or more affordances and the behaviour of each of them is governed by a set of norms. At the level of abstraction in SAME-ML, the details of affordance would not be dealt with and would be left to the implementation level. To reflect the steps in writing the specification, a sequence is established for defining the characteristics of agents, it is: signs, affordances and norms. To specify an agent, there must be a set of signs, at least one affordance and at least one norm.

```
<agent name="  ">
  <signs> ..... </signs>
  <affordance name="" description=""/>
  <affordance name="  " />
  <norm name="  " > ..... </norm>
  ......
</agent>
```

3.3.3 Signs and Radical Subjectivism

One of the essential philosophical stances of the semiotic framework, radical subjectivism, leads to the concept of local perception of an agent where only the signs that are of interest to the agent will be included. The perception is then thought of as a set of facts. The set of signs enumerates all facts that are of interest to the agent. Because it is local, each agent would have one and only one <signs> element defined in it. The <fact> element can take three possible forms: a simple definition that only identifies the fact name and attaches a description; the second and third form involve propagation of fact changes to and from other agents according to our meta-model. A fact can be changed as a result of changes in other agents' facts (<watch>) or it could induce changes to other agents (<watched-by>). These two forms can be mixed together in one single <fact> definition because of cascade propagation.

```
<signs>
  <fact name="A" description="  " />
  <fact name="B" description="  " >
    <watch agent="X"
     fact-name="AX" />
  </fact>
  <fact name="C" description="  " >
    <watched-by agent="Y"
     fact-name="CY" />
  </fact>
  <fact name="D" description="  " >
    <watch agent="Z" fact-name="DZ" />
    <watched-by agent="X"
     fact-name="DX" />
  </fact>
</signs>
```

In the above example, Fact A is a standalone fact. Change of its value will not cause change to other facts and no other fact value changes will change its value. The fact B will have its value changed if the value of fact AX defined in agent X is changed. In other words, agent X is being monitored for its fact AX. Fact C will cause change to fact CY defined in agent Y. Finally, the value of the fact D will be changed as a result of monitoring fact DZ and at the same time, it is being monitored by agent X to induce change to fact DX.

3.3.4 Norms

Format of application norms in SAME-ML is a direct translation of classical semiotic norm. <whenever> tag specifies the context in which this norm is effective and the <condition> determines whether the <action> should be taken and in what way. The type attribute of the <deontic> is restricted to the values of *permitted*, *obliged* and *prohibited*.

```
<norm name="  " >
  <description> ..... </description>
  <whenever> ..... </whenever>
    <condition> ..... </condition>
    <deontic type="obliged" />
    <action name="  " />
```

```
</norm>
```

3.3.5 Environment and Agent Characteristics

According to the meta-model, the environment is a conceptual entity that corresponds to an implementation realisation of a set of software agents being executed within a boundary. The boundary could be a computer or a program depending on the actual implementation. Since the grouping of agents into an environment is an implementation consideration, it should be separated out from the specification of other entities to provide easy maintenance, for example, to produce an alternative implementation solution by regrouping agents without changing the definitions of agents.

```
<environment name="  " type="  ">
  <contain agent="  "
  think-time="  "
  init="  " initi-desc="  " />

  <contain agent="  "
  think-time="  "
  init="  "
  init-desc="  " />
  ......
</environment>
```

Other attributes in the <contain> tag are used to specify other agent run-time parameters. <think-time> determines the responsiveness of an agent is one of these parameters. Furthermore, there may bet some initialization procedures to be done before an agent is instantiated to run. <init> specifies these procedures. Like affordance, these initialization procedures can only be expressed as algorithmic details and should therefore be handled at the lower level of abstraction. It is specified as a name, similar to affordance. The <initialization-desc> provides a description of the initialization procedure for documentation purpose.

4 AN EXAMPLE

4.1 Inter Agent Tracking

The following example models action tracking commonly found in many control and monitoring systems. Consider three agents A, B and C in an environment where B and C would track the action of A. A generates events b and c randomly, and B responds to event b while C responds to c. Events b and c are signs exhibited by A in the environment, but from the viewpoint of B, b is the only sign of interest and the same applies to C. This example was implemented by first identifying the capabilities of the agents as:

- A - firing an event at random intervals in milliseconds (with an odd number of milliseconds resulting in an event b, and an even interval giving an event c);
- B - responding to event b;
- C - responding to event c.

The norms are simple:

- A - if <random interval expires> <obliged> <A> <generate event>
- B - if <event b occurs> <obliged> <B> <print a message>
- C - if <event c occurs> <obliged> <C> <print a message>

The specification of this example is presented in Appendix.

4.2 Code Generation and Implementation

A SAME-ML specification contains all information required to produce an implementation. Scripts developed by tools such as XSLT and XQUERY (Chamberlin 2002; Zisman 2000) can be used to extract information from specifications to generate executable codes. In our implementation, we have developed Java code templates for different types of agents, environments and fact classes. The building of the prototype for the above example by transforming the specifications to implementation codes has been done manually. In the process, code templates for the semiotic agents, their norms and the environment were used. The algorithmic details of the affordances were coded as methods of the corresponding Java agent class derived from the templates. Information required to instantiate a prototype is extracted from the specifications. Norms are translated to rules according to the specifications. Although tools for automatic code generation have not been completed in our work, information defined in the SAME-ML specifications and the code templates are sufficient to support full code generation. By changing the information extraction scripts and the templates, codes for different implementation platform can be produced.

5 CONCLUSIONS

To enable SAM and NAM of the semiotic framework for wider adoption, a modelling language for semiotics based agent systems is defined with the MDD approach used as a reference. Semiotics and the semiotic framework are reviewed to define a meta-model in which semiotic agent is used as the main component. Application of the language to an example of inter-agent tracking shows the adequacy in specifying agent systems. The language has also been used in other work of multimedia systems modelling not reported in this paper. Two key themes are regarded as the core of MDD: raising the level of abstraction of the models to help designers focusing on the problem on hand instead of the programming details and increasing the level of automation in the transformation of the model specification to implementation. The design of SAME-ML fits into these two themes well although further work has to be done to achieve full automation in implementation generation. Further research is being done to investigate the incorporation of other semiotic framework concepts such as the role of agents and their ontological dependence into SAME-ML.

REFERENCES

Andersen, P. B. (1991) Computer semiotics. *Scandinavian Journal of Information Systems, 3*, pp. 3-30.

Barry, C. & Lang, M. (2001) A survey of multimedia and web development techniques and methodology usage. *IEEE Multimedia, 8*(2), pp. 52-60.

Biron, P. V., & Malhotra, A. *XML schema part 2: Datatypes.* http://www.w3.org/TR/xmlschema-1 (last accessed 2009)

Chamberlin, D. (2002) XQuery: An XML query language. *IBM Systems Journal, 41*(4), pp. 597-615.

ITU. (2000) *Z.100 SDL-92 specification and description language.* ITU.

Joslyn, C. & Rocha, L. M. (2000) Towards semiotic agent-based models of socio-technical organizations. *Proceedings of AI, Simulation and Planning in High Autonomy Systems (AIS 2000)Tucson, Arizona, USA.* pp. 70-79.

Liu, K. (2000) *Semiotics in information systems engineering.* Cambridge, U.K., New York, Cambridge University Press. ISBN 0521593352

OMG. *Unified modeling language.* http://www.uml.org/ (last accessed 2009)

Stamper, R. (1994) Social norms in requirements analysis: An outline of MEASUR. In M. Jinotka, J. Goguen & M. Bickerton (Eds.), *Requirements engineering: Social and technical issues.* pp. 107-139 Academic Press Professional Inc., San Diego, CA, USA. ISBN 0-12-385335-4

Stamper, R. (2000) New directions for system analysis and design. *Enterprise information systems.* pp. 14-39 Kluwer Academic Publishers. ISBN 0-7923-6239-X

Thompson, H. S., Beech, D. & Maloney, M. *XML schema part 1: Structure.* http://www.w3.org/TR/xmlschema-1 (last accessed 2009)

Zemanek, H. (1966) Semiotics and programming languages. *Communications of the ACM, 9*(3), pp. 139-143.

Zisman, A. (2000) An overview of XML. *Computing & Control Engineering Journal, 11*(4), pp. 165-167.

APPENDIX

```
<system  name="AgentTracking"
   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
   xmlns='http://xml.netbeans.org/schema/sameMLSchema'
   xsi:schemaLocation='http://xml.netbeans.org/schema/sameMLSchema
       sameMLSchema.xsd'>
   <agent name="A">
   <signs>
       <fact name="RandomPeriod" />
       <fact name="Event">
           <watched-by agent="B" fact-name="Event" />
           <watched-by agent="C" fact-name="Event" />
       </fact>
   </signs>
   <affordance name="FireEvent" />
   <affordance name="GetRandomPeriod" />
   <norm name="Fire Event">
       <description>Create event</description>
       <whenever>running</whenever>
       <condition>random-period-expires</condition>
       <deontic type="obliged" />
       <action name="FireEvent" />
       <action name="GetRandomPeriod" />
   </norm>
   </agent>
   <agent name="B">
   <signs>
       <fact name="Event">
           <watch agent="A" fact-name="Event" />
       </fact>
   </signs>
   <affordance name="Respond" />
   <norm name="Repond to Event">
       <whenever>running</whenever>
       <condition>event occurs</condition>
       <deontic type="obliged" />
       <action name="Respond" />
   </norm>
   </agent>
   <agent name="C">
   <signs>
       <fact name="Event">
           <watch agent="A" fact-name="Event" />
       </fact>
   </signs>
   <affordance name="Respond" />
   <norm name="Repond to Event">
       <whenever>running</whenever>
       <condition>event occurs</condition>
       <deontic type="obliged" />
       <action name="Respond" />
   </norm>
   </agent>
   <environment name="main" type="application">
       <contain agent="A" think-time="500" init="none" />
       <contain agent="B" think-time="500" init="none" />
       <contain agent="C" think-time="500" init="none" />
   </environment>
</system>
```