A Decade of Software Design and Modeling: A Survey to Uncover Trends of the Practice

Suresh Kumar Mukhiya

Western Norway University of Applied Sciences

PCS 953 / DAT 353
Spring 2019
Bergen
Norway

Outline

Background
- Software Engineering, SE

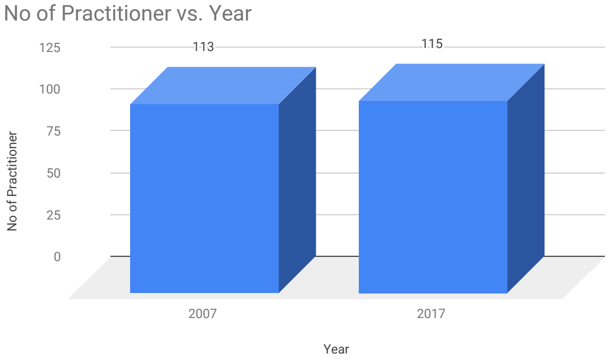
Modeling
- Model
- Modeling Languages

Meta-Modelling

Summary

Survey conducted on two phases

- with 228 software paractitioner
- April-December, 2007
- March-November, 2017



Goal of the Survey

- Uncover **trends in the practice** of `software design` and **adaptation pattern** of `modeling language`

Goal of the Survey

- Uncover **trends in the practice** of `software design` and **adaptation pattern** of `modeling language`

Software Engineering

- Software covers two aspects:
  - **Structural**, data and knowledge representation
  - **Procedural**, description of dynamically changes of data
- Traditional view of software development:
  $Algorithms + DataStructures = Programs$
- MDE view of software development:
  $Models + Transformations = Software$

Data representation

- A conceptual model is an abstract representation of a system, made of the composition of concepts which are used to help people know, understand, or simulate a subject the model represents.
- How can we formalize this?
  - How can we represent concepts?
  - How can we represent relations between concepts?

Modelling Perspectives

Software models represents different perspectives of the system:

- External perspective, models the context of the system, or the interfaces to other systems
- Interaction perspective, models the interaction between the system and its environment
- Structural perspective, models the structure of the system (or the data)
- Behavioral perspective, models the dynamic behavior of the system

Formalization

To reason about the domain we need a formalization that have:

- Enough expressive power:
  - To express relations between model types
  - To express relations between model types and instances
- The right level of abstraction
- Possibility to be visualized (both machine and human understandable)

Sets

Sets can be used to represent concepts. Where elements of a set represents instances of a concept. To state that an individual is element of a set we use $\in$ notation e.g. to say that Yngve is a professor we write $yngve \in Professor$

Example of concepts from the University domain:

- Students, Professor, Subject, Courses etc.

Expressibility of Sets

How can we relate concepts?

Functions (and relations) are traditional ways of relating sets.

Function ensures that for each input of the source set it exists an unique output of the target set. Hence a function f can be visualized with an array $f : X \to Y$. Illustrating for each $x \in X$ it exists an unique $y \in Y$

Expressibility of functions:

- Functions can be used to model isA relations. E.g. to model that all students is a person we use an inclusion function: $\iota : Student \to Person$ alternative notation $Student \subset Person$
- Functions can be used to merge/classify elements e.g. E.g. we can make a function $Year : Students \to \{1, 2, 3, 4, \bullet\}$ To model how long a PhD is in the study, where $\bullet$ models students that use more than 4 years.

Special sets

- It exists a set with one unique mapping to every other set, the emptyset $\emptyset$. A set that has a unique mapping to every other sets is called an initial set (initial object)
- It also exists sets that have a unique mapping from every other set, a one point set $\{\bullet\}$. Sets (objects) satisfying this properties are called terminal. All elements of a set X is identified in $\{\bullet\}$ by the unique mapping $x : X \to \{\bullet\}$
- An element (constant) c in a set X is represented by a function from $c : \{\bullet\} \to X$

How to merge sets?
Union, intersection disjoint union etc..

Relations

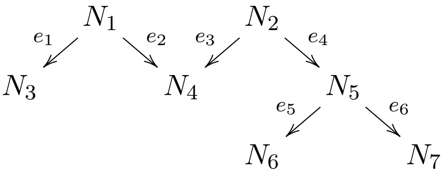
How to relate students and courses?
We can not use functions since a student might take several courses and a course has usually many students.
A relation is a subset of the cartesian product: For example Enrolment could be expressed as a binary relation between Courses and Students $Enrolment \subset Student \times Course$
We can also think about the relation as predicate on the combined types that is true for the pairs in the relation.
We can model that a $Student$ is a $Person$ by making a unary relation $Student \subset Person$, in other words $Student$ is a predicate on $Person$ that is true for all persons that are students.

Graphs

A (directed) **graph** G is defined by $G = (N_G, E_G, src_G, trg_G)$ where

- N_G is a set of nodes
- E_G is a set of edges
- Function, $src_G : E_G \to N_G$, returns the source node of a edge
- Function, $trg_G : E_G \to N_G$, returns the target node of a edge

Example



Example Graph

The graph $G = (N_G, E_G, src_G, trg_G)$ is given by the following digram:

$$
\begin{tikzcd}
 & N_1 \arrow[dl, "e_1"'] \arrow[dr, "e_2"] & & N_2 \arrow[dl, "e_3"'] \arrow[dr, "e_4"] & & \\
N_3 & & N_4 & & N_5 \arrow[dl, "e_5"'] \arrow[dr, "e_6"] & \\
 & & & N_6 & & N_7
\end{tikzcd}
$$

- $N_G = \{N_1, N_2, N_3, N_4, N_5, N_6, N_7\}$
- $E_G = \{e_1, e_2, e_3, e_4, e_5, e_6\}$
- $src_G(e_1) = N_1 = src_G(e_2), src_G(e_3) = N_2 = src_G(e_4), src_G(e_5) = N_5 = src_G(e_6)$
- $trg_G : G(e_1) = N_3, trg_G(e_2) = N_4 = trg_G(e_3), trg_G(e_4) = N_5, trg_G(e_5) = N_6, trg_G(e_6) = N_7$

Graph Homomorphism

A **graph homomorphism** $\phi : G \to H$, between to graphs G, H is defined by two mappings $\phi_N : N_G \to N_H$ and $\phi_E : E_G \to E_H$, that preserves source and target.
It means that for each edge $e \in E_G$ we have that:
$src_H(\phi_E(e)) = \phi_N(src_G(e))$ and $trg_H(\phi_E(e)) = \phi_N(trg_G(e))$
Is there a graph homomorphism from the graph H given by the following digram:

$$A_1 \xrightarrow{f_1} A_2 \xrightarrow{f_2} N_3$$

And the graph G?

$$\begin{tikzcd}
 & N_1 \arrow[dl, "e_1"'] \arrow[dr, "e_2"] & & N_2 \arrow[dl, "e_3"'] \arrow[dr, "e_4"] & & \\
N_3 & & N_4 & & N_5 \arrow[dl, "e_5"'] \arrow[dr, "e_6"] & \\
 & & & N_6 & & N_7
\end{tikzcd}$$

Example Graph Homomorphism

It's actually two graph homomorphisms ϕ and ρ from H to G:

- ϕ: $\phi_N(A_1) = N_2, \phi_N(A_2) = N5, \phi_N(A_3) = N_6$ and $\phi_E(f_1) = e_4, \phi_E(f_2) = e5$
- ρ: $\rho_N(A_1) = N_2, \rho_N(A_2) = N5, \rho_N(A_3) = N_7$ and $\rho_E(f_1) = e_4, \rho_E(f_2) = e6$

$$A_1 \xrightarrow{f_1} A_2 \xrightarrow{f_2} N_3$$

And the graph G?

$$\begin{tikzcd}
 & N_1 \arrow[dl, "e_1"'] \arrow[dr, "e_2"] & & N_2 \arrow[dl, "e_3"'] \arrow[dr, "e_4"] & & \\
N_3 & & N_4 & & N_5 \arrow[dl, "e_5"'] \arrow[dr, "e_6"] & \\
 & & & N_6 & & N_7
\end{tikzcd}$$

Graph based modelling

Nodes represents concepts and edges represents relations between concepts:

Concepts

- Person
- Employee
- Customer

Example

$$
\begin{tikzcd}
 & Person \arrow[dl] \arrow[dr] & \\
Employee & & Customer
\end{tikzcd}
$$

More concepts and relations

Note that we have different types of relations:

- An Employee isA Person
- A Customer books a room (formally a RoomType)

Example

$$\begin{tikzcd}
& Person \arrow[dl] \arrow[dr] & & Booking \arrow[dl] \arrow[dr] & & \\
Employee & & Customer & & RoomType \arrow[dl] \arrow[dr] & \\
& & & Double & & Single
\end{tikzcd}$$

Typing and instance

An instance of a graph G is a graph I together with a graph homomorphism $\iota : I \to G$, we say that I is typed by G

Typing

- Person typed by Class
- childOf typed by Association
- 2..2 typed by Property

An instance conforms to a model (graph) if it satisfies all constraints of the model. A person instance will conform to the model above if it has exactly two parents

Modelling

Model, from Latin *modulus* meaning measure/standard

- A model is used in two ways either as a:

  Prescription model (usually a miniature) used to represent a system before creation, as a pattern for design

  Description model used to represent some major aspects of an item, object, system, or concept

- A model need to meet the following 3 criteria:

  Reflection The model reflects some properties of the original (system)

  Abstraction The model describes only some of the "interesting" properties of the original

  Substitution The model can be used instead of the original for some purposes

Modelling Abstraction

- Abstraction key element in modelling used for:

  Generalization generalize specific features of model elements

  Classification classify model elements into coherent clusters

  Aggregation Aggregate model elements into more complex ones

  Information hiding Hide implementation details from presentation

Modeling and development methods

Software models are used for different purposes in different development methods:

- Agile methods, models are means to facilitate discussion about systems or design
- Structured methods (UP), models are mainly used for documentation and specification
- MDE, models are used for system development and code generation

General Purpose Modeling Languages (GPML)

- General purpose modelling Languages (GPML) are made to represent some aspects of a system (object of study)
- GPMLs are often graph based with limited support for constraints
  - In practice: UML+ OCL
- The semantics is often not precise or undocumented
- GPMLs are often made to support a specific technology
  - UML object orientation
  - ER relational databases

Some modeling languages

Behavior Trees formal, graphical modelling language to unambiguously represent natural language requirements for large-scale software-integrated systems

Business Process Modelling Notation (BPMN, it's XML form BPML and executable form BPEL) examples of a Process Modelling language

Object Role Modelling (ORM) conceptual DB modelling language

Petri nets For model checking, graphically-oriented simulation, and software verification

Unified Modelling Language (UML), diagrams for behaviour and structure specification

Formal languages Algebraic, logic, categorical, set based, ...

Unified Modeling Language (UML)

- UML is defacto industry standard for modelling
- UML consists of 13 different diagrams, most important are:
  - Activity diagrams, activities in processes
  - Use case diagrams, interaction between system and environment
  - Sequence diagrams, interaction between system components (and actors)
  - Class diagrams, structure of the system
  - State diagrams, system dynamics

Some facts about UML

- UML can only express constraints on binary relations, basically UML specifies only cardinality constraints
- UML has serious issues regarding:
  - Semantics; UML models may be ambiguous and have semi-formal semantics
  - Complexity; UML uses 13 different types of diagrams
  - Expressibility; To express constraints over higher order relations one need to use string based logic, (Object Constraint Language, OCL)

Formal modeling languages

Pros:

- Nice semantics, several fundamental Software Engineering problems solved by use of formal methods
- Set based semantics (logic, algebras, type theory, . . .)

Cons:

- No concept of meta-modelling
- Hard for software engineers to apply in practice
- Lack of good software development tools based on formal approaches
- Only used by well trained experts
- High cost low productivity

Domain Specific Modelling Languages DSMLs

- Domain Specific Modelling Languages DSMLs are modelling languages made for a specific domain
- In MDE:
  - DSMLs usually specified by a graph-based meta-model + text-based Constraints
  - In practice: UML, UML profiles + OCL

Domain Specific Modelling Languages DSMLs

- Domain Specific Modelling Languages DSMLs are modelling languages made for a specific domain
- In MDE:
  - DSMLs usually specified by a graph-based meta-model + text-based Constraints
  - In practice: UML, UML profiles + OCL
- Problem
  - Typing and constraints are specified by different languages (having different meta-models)
  - Model transformations usually not constraint-aware
- Solution
  - Diagrammatic specification formalism where the meta-modelling considers both typing and constraints

Meta Modelling

- A meta-model is a model of a modelling language
- Meta-models are used to define modelling languages
- E.g. in OO modelling a person is an instance of class
- Meta-modelling is used to create Domain Specific Modelling Languages DSMLs, i.e. one create language constructs for important domain concepts, e.g. a student and a teacher is instances of persons

Meta-model example

- Models: first class entities

Meta-model example

- Models: specified by means of a modelling language

Meta-model example

- Modelling language: corresponding meta-model + semi-formal semantics

OMG Meta-modelling Levels

OMG levels	**OMG Standards/examples**
M_3: Meta-meta-model	MOF
M_2: Meta-model	UML language
M_1: Model	A UML model: Class "Person" with attributes "name" and "address"
M_0: Instance	An instance of "Person": "Ola Nordmann" living in "Sotraveien 1, Bergen"

MOF based modelling languages

UML System on a Chip for microchip/hardware/firmware/software definition

SoaML for service-oriented architecture

Business Process Modelling Notation (BPMN, together with it's XML form BPML and executable form BPEL) examples of a Process Modelling language

SysML for modelling large, complex systems of software, hardware, facilities, people and processes

UPDM for modelling enterprise architectures

CWM for data warehouse

Benefits of MDE

- Engineers can reason about the system at different abstraction levels
- Platform independent models without concern of implementation details
- Less errors and faster development speed by automatic software generation
- Software adoption by (automatic) model transformations

Chalenges in MDE

- Modelling languages need to haver the right abstractions, i.e. one need domain specific modelling languages
- Specification of constraints integrated in the meta-modelling approach, i.e. graphical modelling formalisms with well defined semantics
- MDE traditionally concerned by software architecture and behaviour, need to have technologies for:
  - Model management (version control, meta-model evolution)
  - Model based security engineering
  - Model based testing, software dependencies, ...

State of the art in MDE

Modeling UML or EMF used as modelling language

Model transformations Rule based (e.g. Atlas) or ad hoc transformations are used

Meta modelling Only tool support for 2 levels of meta-modelling

Tool support Eclipse based (EMF, GMF) tools

Software constraints Specified in text based language (OCL)

Links to resources

- mde-model-driven-engineering-reference-guide
- model-driven-engineering
- mda-model-driven-architecture-basic-concepts
- Diagram-Predicate-Framework
- Eclipse-Modeling-Technologies

Litterateur

- Conceptual data modelling from a categorical perspective. ter Hofstede, A. H., Lippe, E. and Frederiks, P. J. M. (1996), The Computer Journal, 39(3), 215-231.
- View updates in a semantic data modelling paradigm. Johnson, M., Rosebrugh, R. and Dampney C.N. G. In: Proceedings of the 12th Australasian database conference. IEEE Computer Society, 2001. p. 29-36
- Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce. Fensel D. Berlin: Spring-Verlag
- Symbolic graphs for attributed graph constraints. Journal of Symbolic Computation, 46(3), 294-315. Orejas, F. (2011)
- Diagram predicate framework: a formal approach to MDE. Rutle, A. (2010)

More Litterateur

- Domain-Specific Languages, Martin Fowler (With Rebecca Parsons), Addison Wesley
- Prolog-based infrastructure for RDF: performance and scalability. Wielemaker, J., Schreiber, A.T., Wielinga, B.J. In: (Ed.), The Semantic Web - Proceedings ISWC'03, Sanibel Island, Florida (pp. 644-658). Springer Verlag
- Alloy: a lightweight object modelling notation. Jackson, D. (2002). ACM Transactions on Software Engineering and Methodology (TOSEM), 11(2), 256-290