MODEL DRIVEN SOFTWARE DEVELOPMENT

LECTURE : I

TEXT BOOK AND REFERENCE MATERIAL

Textbook(s):

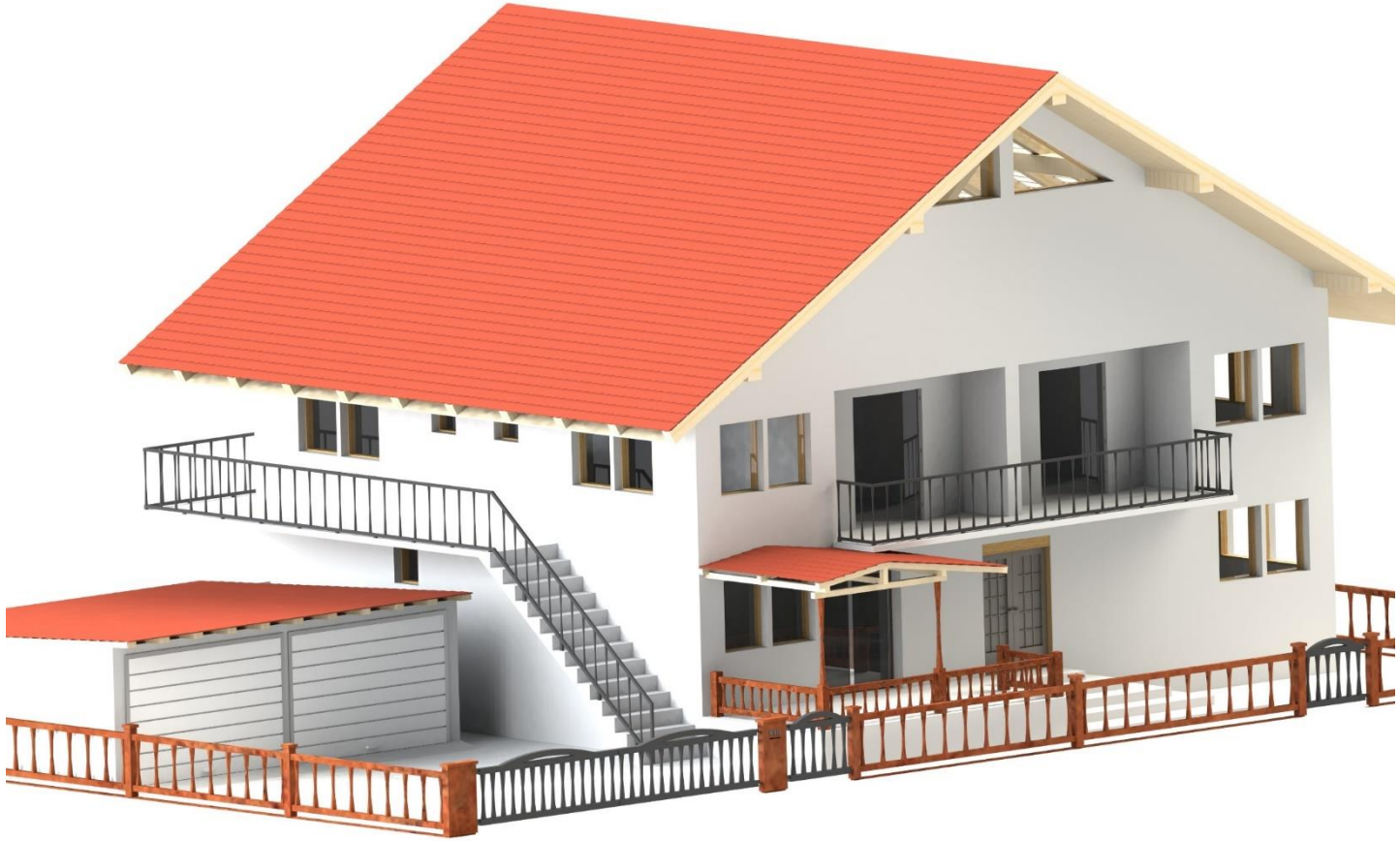
Model-Driven Software Development: Technology, Engineering, Management by Thomas Stahl, Markus Voelter and Krzysztof Czarnecki, Publisher: Wiley; 1st Edition (May 19, 2006). ISBN-10:0470025700

Reference Material:

- Model-Driven Software Engineering in Practice by Marco Brambilla, Jordi Cabot and Manuel Wimmer, Morgan & Claypool Publishers; 1st Edition (September 26, 2012). ISBN-10: 1608458822
- The Pragmatic Programmer: From Journeyman to Master by Andrew Hunt and David Thomas, Addison-Wesley Professional; 1st Edition (October 30, 1999). ISBN-10: 020161622X
- Model-Driven Software Development: Integrating Quality Assurance by JorgRech and Christian Bunse, Information Science Reference; 1st Edition (August 22, 2008). ISBN-10: 160566006X
- Model-Driven Software Development with UML and Java by K. Lano, Course Technology (August 15, 2009). ISBN-10: 1844809528

MODEL

What is model?



MODEL

- "A model is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the actual system."

(Jean Bézivin)

- "Models help in developing artefacts by providing information about the consequences of building those artefacts before they are actually made."

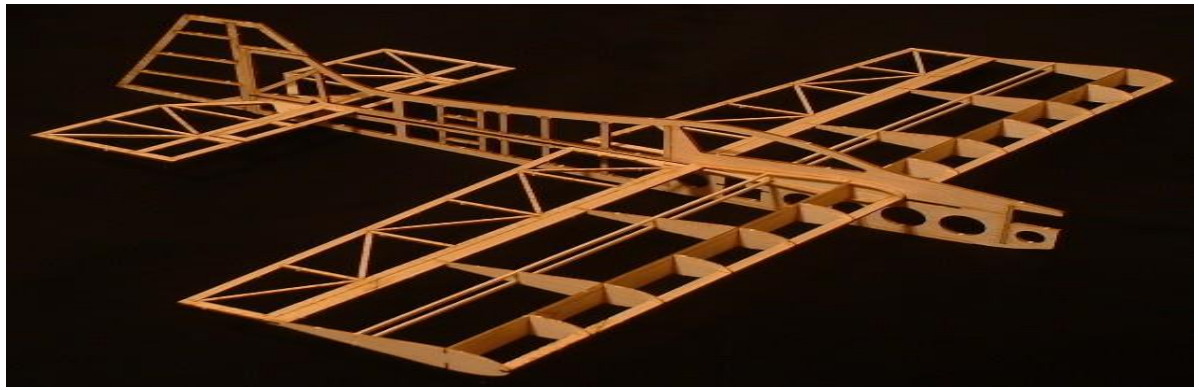
(Ludewig)

- "A model of a system is a description or specification of that system and its environment for some certain purpose."

(OMG)

MODELS IN TRADITIONAL ENGINEERING

Models used in all branches of engineering



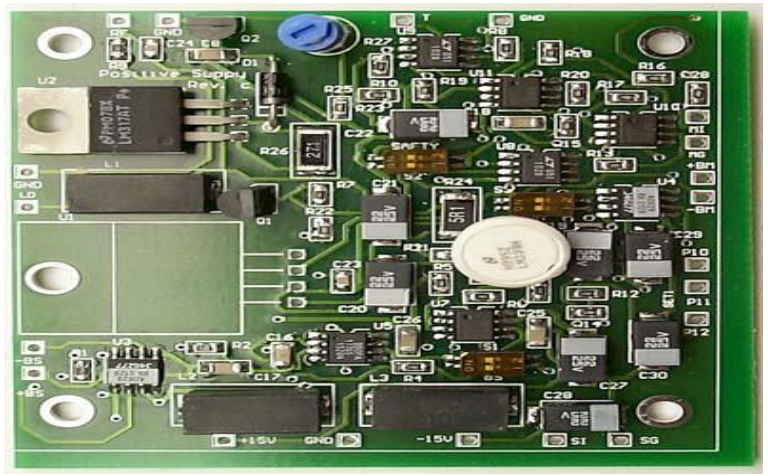
MODEL

- Abstract representation of a system's structure, function or behavior
- Simplification, description and specification of system
- Can answer questions regarding actual System (Analysis, prediction, inferences)
- Provide purpose, understanding and risk analysis of system
- Model allows
  - Discard irrelevant details

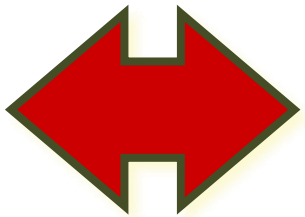
Table → **Abstraction** → Rectangle, Four legs, Wooden, Clean design → **Model**

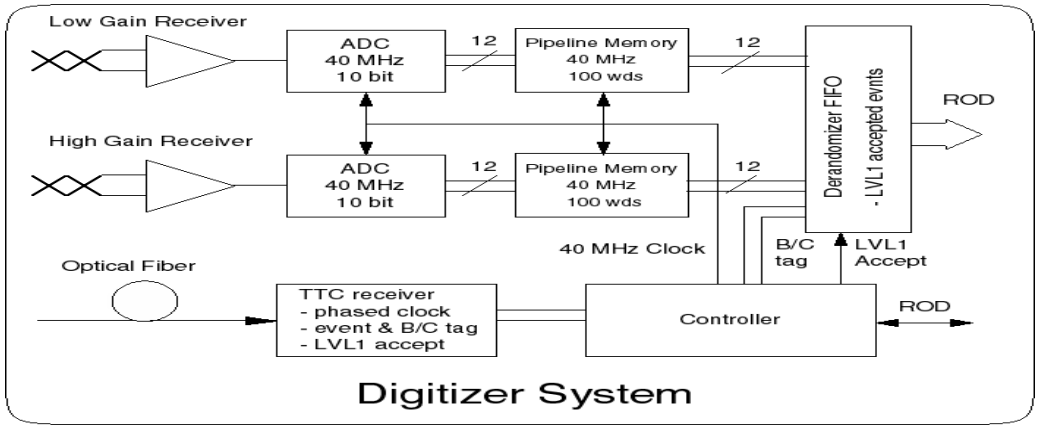
WHAT IS A MODEL?

- Some definitions:
  - A simplified representation used to explain the workings of a real world system or event.
  - A reduced/abstract representation of some system that highlights the properties of interest from a given viewpoint. The viewpoint defines concern, scope and detail level of the model.



Modeled system

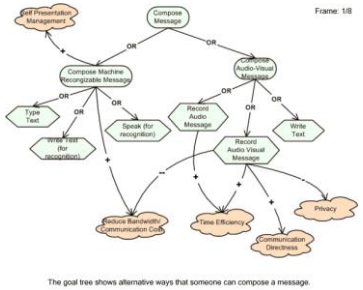




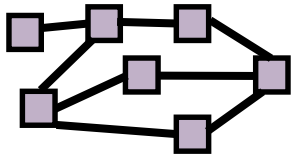
Functional model

MODEL-DRIVEN SOFTWARE DEVELOPMENT

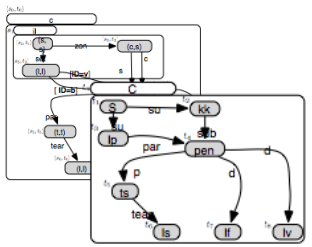
- Model: It's an *abstract* description of *software artifact*, created for a *purpose*.



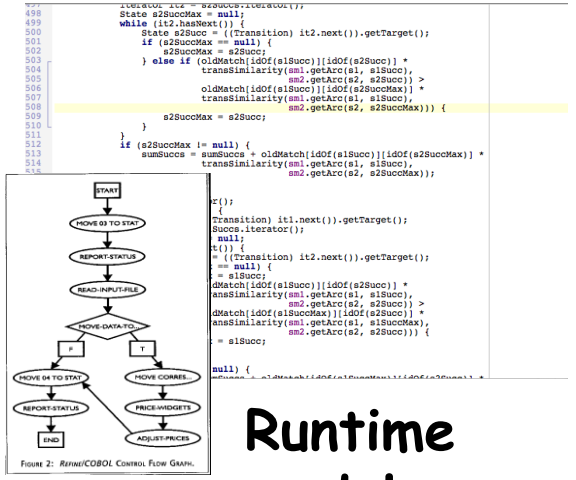
Requirements models



Architecture models



Behavioural models



Runtime models

- **Why is it abstract?**
  - Things are omitted, e.g., detail, views, subsystems
- **What purposes?**
  - Analysis, test automation, code generation, review, explanation, documentation ...

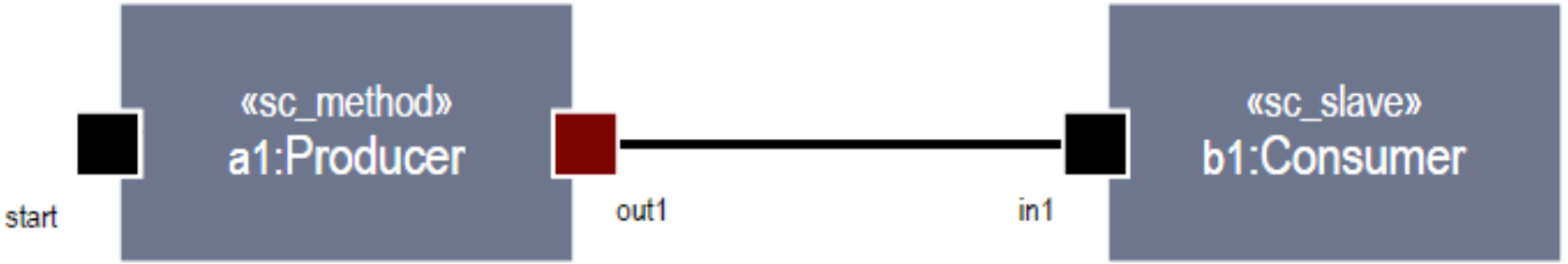
EXAMPLE CODE

```
SC_MODULE(producer)
{
sc_outmaster<int> out1;
sc_in<bool> start; // kick-start
void generate_data ()
{
for(int i =0; i <10; i++) {
out1 =i ; //to invoke slave;}
}
SC_CTOR(producer)
{
SC_METHOD(generate_data);
sensitive << start;}};
SC_MODULE(consumer)
{
sc_inslave<int> in1;
int sum; // state variable
void accumulate (){
sum += in1;
cout << "Sum = " << sum << endl;}
```

```
SC_CTOR(consumer)
{
SC_SLAVE(accumulate, in1);
sum = 0; // initialize
};
SC_MODULE(top) // container
{
producer *A1;
consumer *B1;
sc_link_mp<int> link1;
SC_CTOR(top)
{
A1 = new producer("A1");
A1.out1(link1);
B1 = new consumer("B1");
B1.in1(link1);}};
```

Can you see what this program is about?

CORRESPONDING UML MODEL



Can you see it now?

MODEL IS ABOUT?

Structure

- Data
- Architecture
- Components
- User Interface
- Any aspect of system

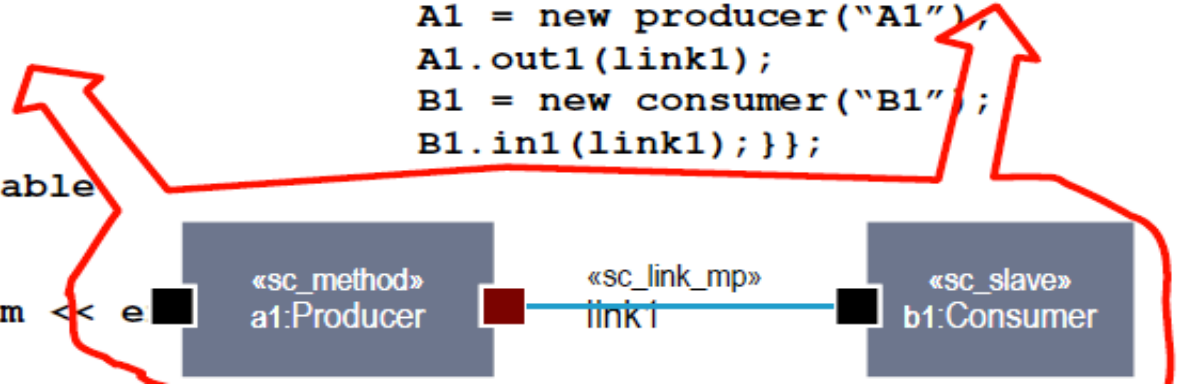
Behavior

- Configurations
- Connections
- Communication
- Business Process

THE PROGRAM & ITS MODEL

```
SC_MODULE(producer)
{
sc_outmaster<int> out1;
sc_in<bool> start; // kick-start
void generate_data ()
{
for(int i =0; i <10; i++) {
out1 =i ; //to invoke slave;}
}
SC_CTOR(producer)
{
SC_METHOD(generate_data);
sensitive << start;}};
SC_MODULE(consumer)
{
sc_inslave<int> in1;
int sum; // state variable
void accumulate (){
sum += in1;
cout << "Sum = " << sum << e
```

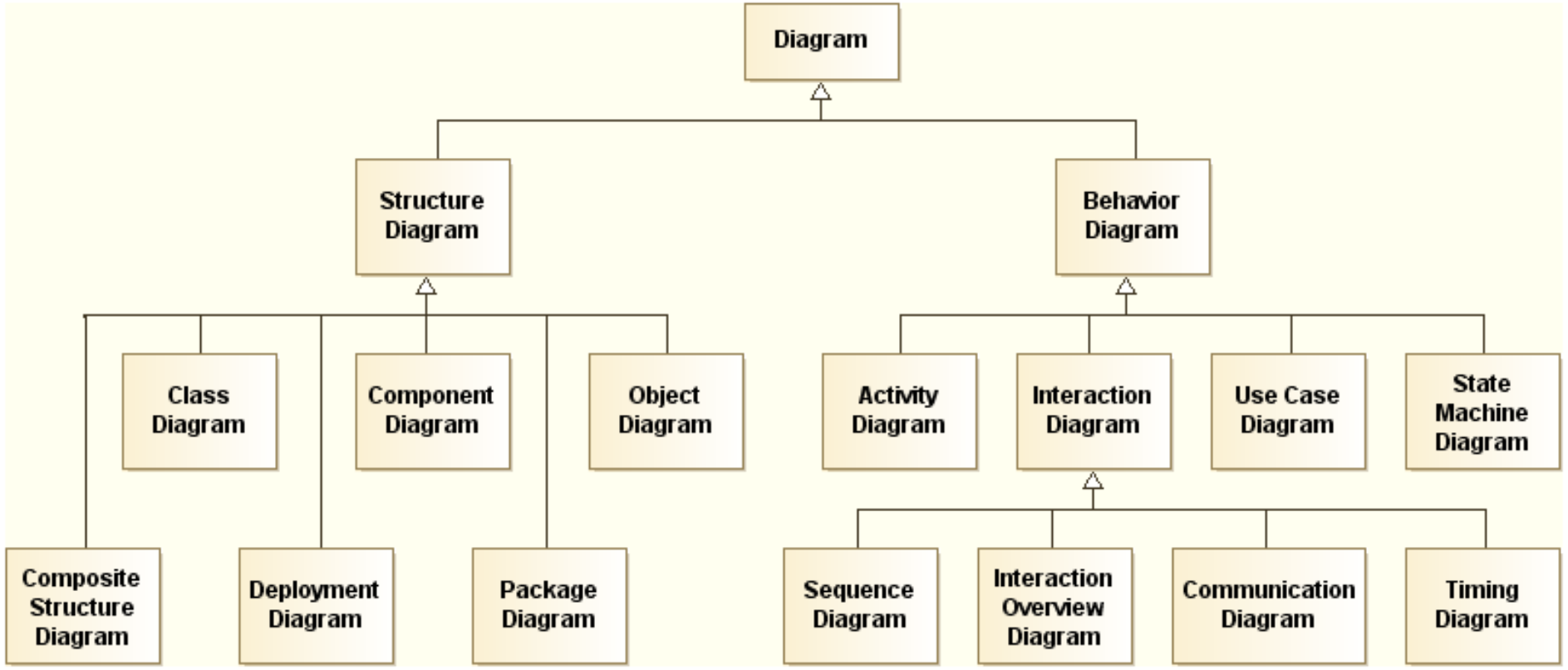
```
SC_CTOR(consumer)
{
SC_SLAVE(accumulate, in1);
sum = 0; // initialize
};
SC_MODULE(top) // container
{
producer *A1;
consumer *B1;
sc_link_mp<int> link1;
SC_CTOR(top)
{
A1 = new producer("A1");
A1.out1(link1);
B1 = new consumer("B1");
B1.in1(link1);}};
```



THE OBJECT MANAGEMENT GROUP (OMG)

- An open membership and non-profit consortium
  - IBM, Microsoft, etc.
- Originally aimed at setting standards for distributed object-oriented systems, and is now focused on modeling (programs, systems and business processes) and model-based standards.

OMG'S UNIFIED MODELING LANGUAGE (UML)



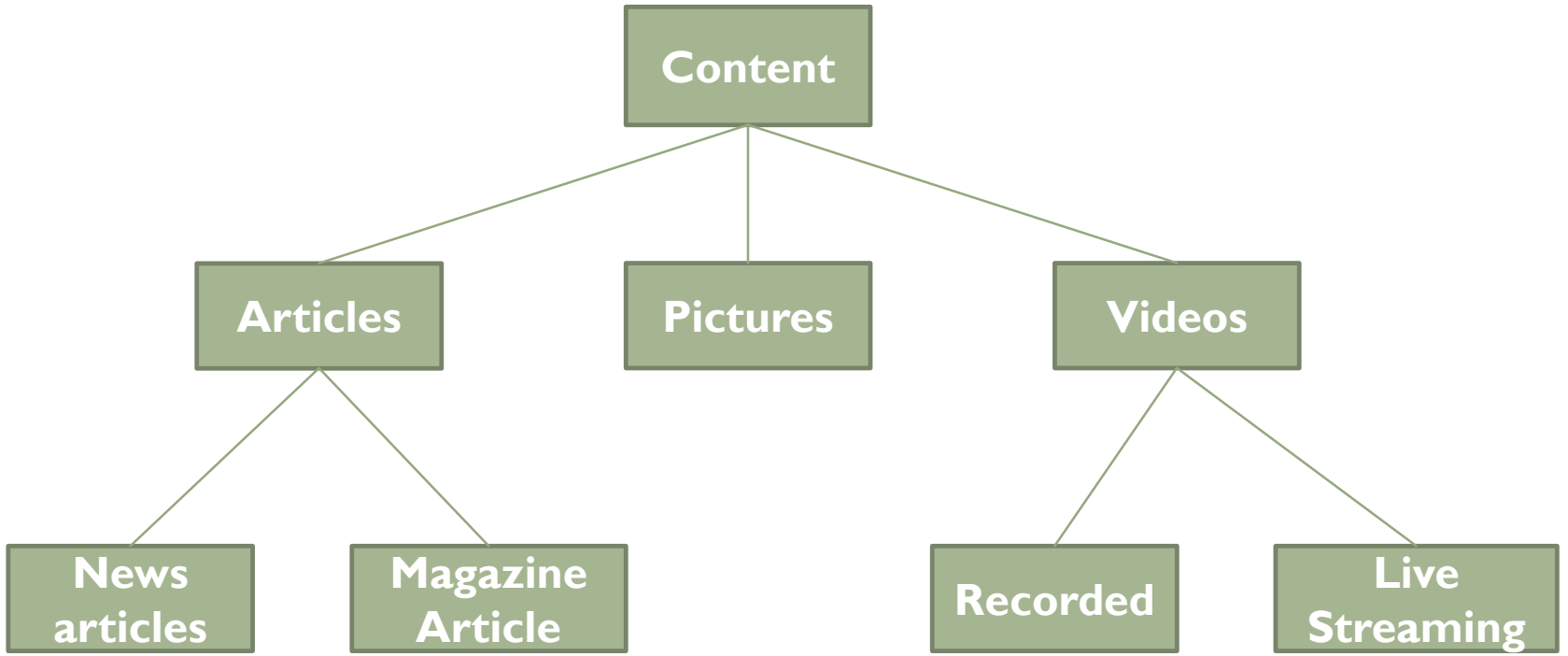
MODEL AS A UML DIAGRAM

Abstraction?

Simplification?

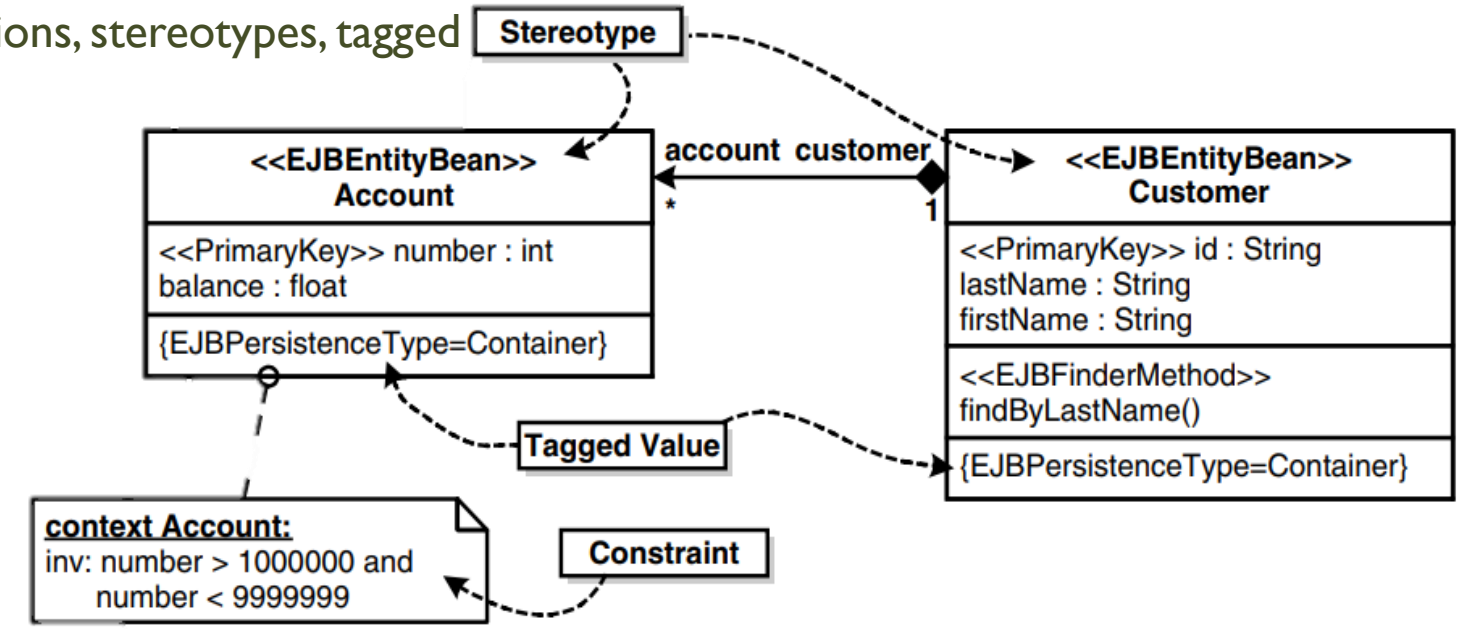
Analysis?

Understanding?



UML PROFILE

- UML Profile contains language concepts that are defined via basic UML constructs such as classes and associations, stereotypes, tagged values, and modeling rules
- Conceptually, a model is an 'instance' of a meta model
- UML meta model contains elements such as Class, Operation, Attribute, or Association
- The meta model concept is one of the most significant concepts in the context of MDSD



- Many commercial modeling tools (e.g., Rational Software Architect, Magicdraw)
- Even an open source UML tools (e.g., Papyrus, TopCased)

MODEL DRIVEN DEVELOPMENT (MDD)

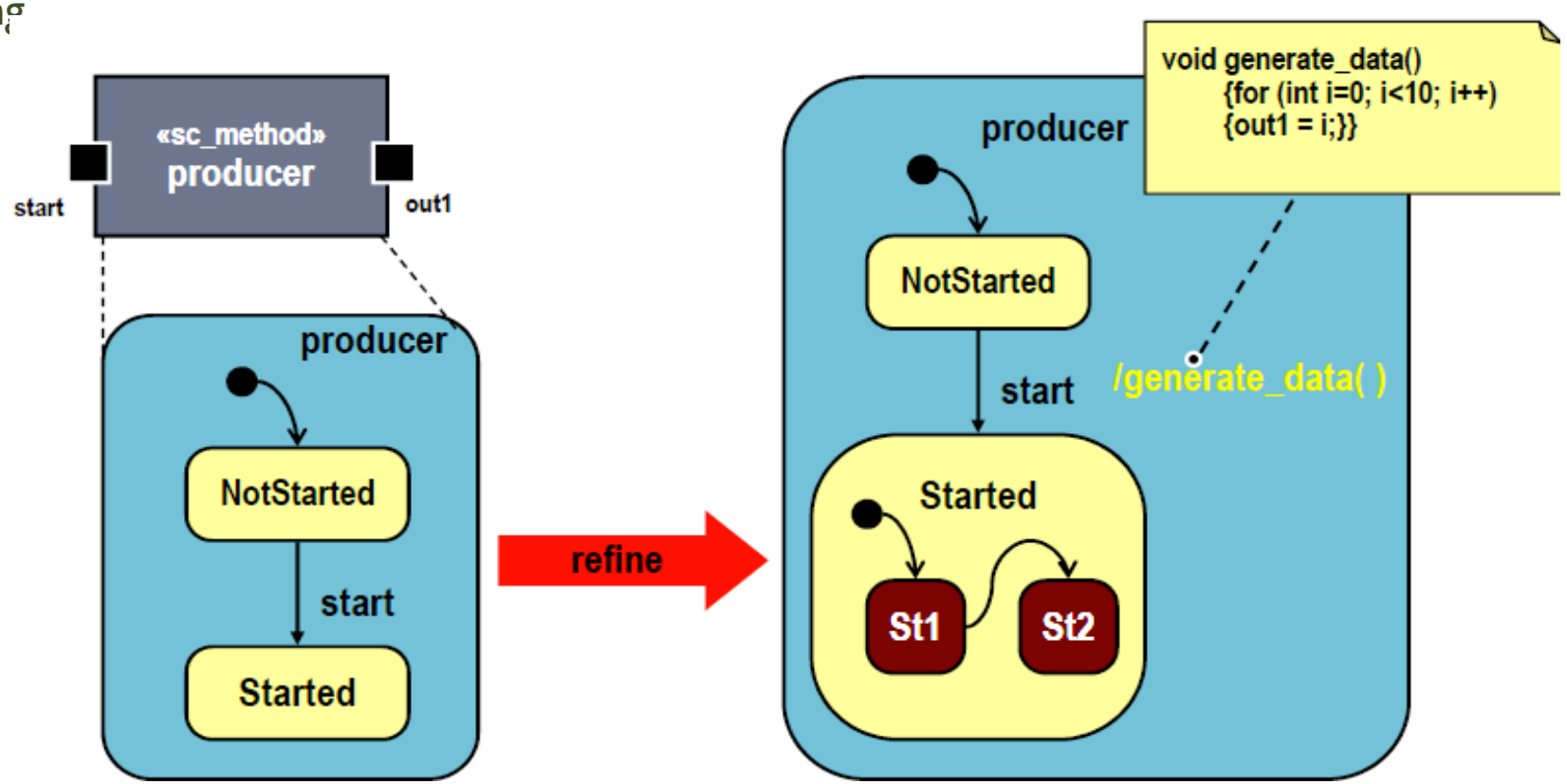
The usage of **M**odels as the main artefacts to **D**rive the software **D**evelopment

MDD --- Models, Transformations, Development processes

Models + Transformations = Software

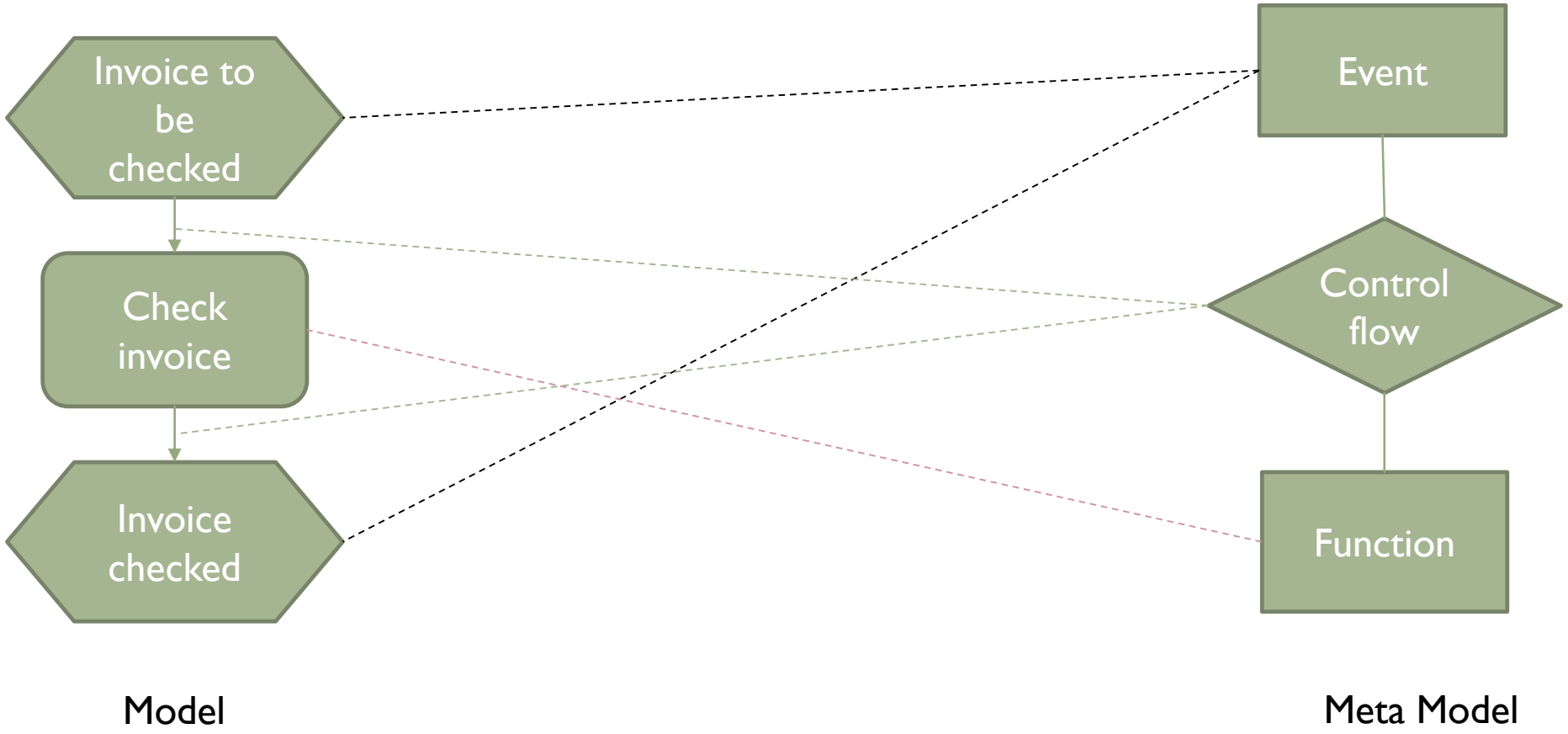
MODERN MBS DEVELOPMENT STYLE

- Models can be refined continuously until the application is fully specified, i.e., the model becomes the system that it was modeling



META MODEL

- Model describing model is called meta model



TRANSFORMATIONS

- MDSD provides appropriate languages for defining model transformation rules
- Rules can be written manually from scratch by a developer, or can be defined as a refined specification of an existing one
- Alternatively, transformations themselves can be produced automatically out of some higher level mapping rules between models
  - Defining a mapping between elements of a model to elements to another one (model mapping or model weaving)
  - Automating the generation of the actual transformation rules through a system that receives as input the two model definitions and the mapping
- Transformations themselves can be seen as models!

BENEFITS OF MDSD

- **MDSD increase your development speed**

  Automation: runnable code can be generated from formal models using transformation steps

- **Cross-cutting implementation aspects can be changed in one place**

  In transformation rules

  Fixing bugs in generated code

- **Reduce Complexity**

  Due to abstractness

- **Reusability**

  Once they have been defined. Architectures, modeling languages and transformations can be used in the sense of a software production line for the manufacture of diverse software systems