Atom Free IT

Domain-oriented modeling with MuDForM

Turning art into engineering

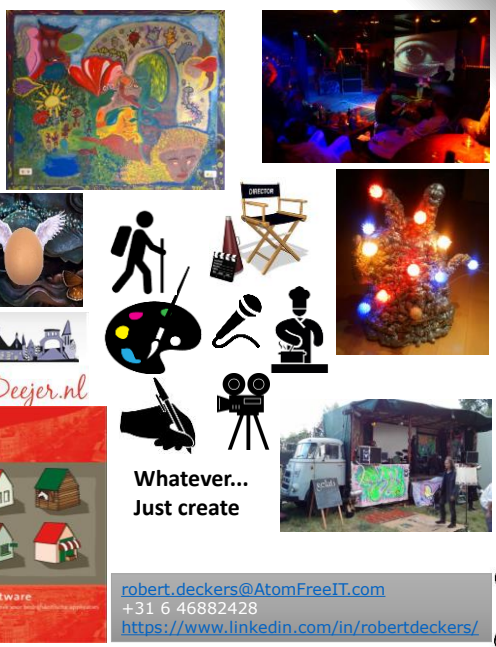
Robert Deckers, in cooperation with Jennek Geels and Niko Stotz

V7 20220223

1

Robert in a nutshell

- **Expertise, focus:**
  Modelling, Model Driven Development,
  Method engineering,
  (Non-functional) Requirements,
  Architecture:
  Aspect oriented design (patterns)
  Research, innovation
  System, software, information, enterprise
  Coaching, consultancy, training
- **Some customers**:
  Currently: Canon, TU/e, HTI.
  UT, UvA, COA, BMW group, KvK, Ohra, Rabobank, NXP, UVIT, CRV, SNS Reaal, Delta Lloyd, Interpolis, APG, NS, Océ, VBI, Shell, ECT, Fortis, Vlisco
- **Jobs:**
  Conscription -1994,
  KISS b.v. –1999,
  Philips Research -2006, Philips Healthcare -2007,
  Sogeti -2013,
  Atom Free IT, PhD VU university -today
- **Education:**
  TU/e informatica -1991,
  PDEng Software Technology -1993

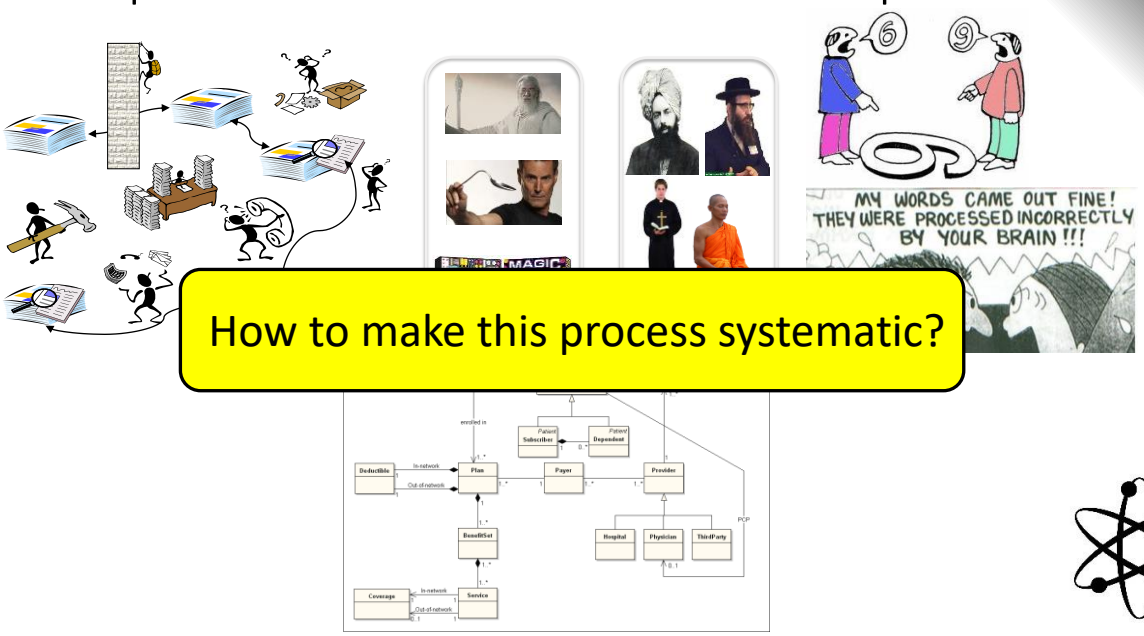


robert.deckers@AtomFreeIT.com
+31 6 46882428
https://www.linkedin.com/in/robertdeckers/

2

2

Capture and define relevant concepts



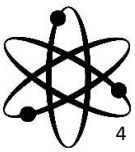
3

Some issues with the current practice

Existing approaches for domain modeling

- are not methods:
  - No detailed steps and guidelines to guide the modelling activity.
  - Often just a notation (and sometimes a metamodel).
- do not see behavior as first class citizen, but are data/state oriented.
- do not explicitly link to human communication (in natural language).
- do not (help to) separate application/feature knowledge from domain knowledge.
- do not offer support making specifications with the domain model.
- do not explicitly support multi-domain and multi-feature aspects.

(These are conclusions from a systematic literature review)



4

MuDForM (Multi-Domain Formalization Method)

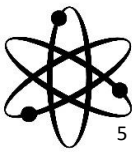
To enable the people involved in a development process to communicate and reason close to their area of knowledge,

we are developing a method to
formalize and integrate knowledge of multiple domains and features

into domain models,

specifications in terms of those domain models,
and integrations of those.

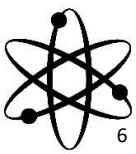
Today's focus



5

MuDForM principles (1)

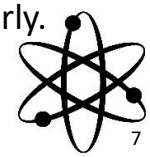
- An engineered method with explicit ingredients:
  underlying model, notation, steps, guidelines.
- Use natural language as entry point.
  Everyone uses it and it has evolved over thousands of years.
- Use engineering guidelines/rules/criteria.
  Go further than underlining nouns, much further.
- Separate domain knowledge, i.e., what can happen/exist
  from feature/system knowledge, i.e., what shall happen/exist.



6

MuDForM principles (2): divide and conquer

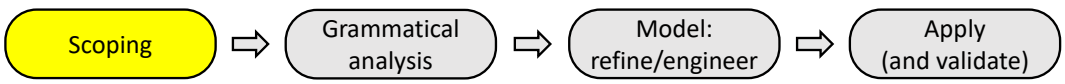
- Do not bother everyone with everything:
  multiple domains/features (with explicit relations between them)
- There is no first time right: first explore and then engineer
  go deep and wide in the beginning and then add precision.
- Focus and manageable: explicitly scope the model and check regularly.

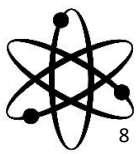


7

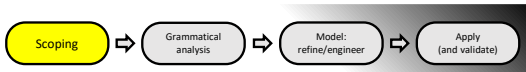
The MuDForM modeling process

From heads and documents to tacit engineered knowledge



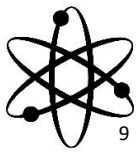


8

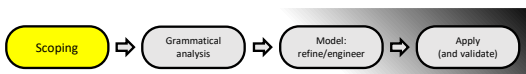


Define scope

- Define the purpose of the model:
  - What do you want to specifiy with the model?
  - What applications/systems do you want build from the model?
- Demarcate the area:
  - List top-of-mind concepts:
    - One minute brainstorm with domain expert.
    - With a group: model storming.
  - What is not in the model:
    - “Adjacent” domains.
    - Other perspectives, for example: sitting on a chair vs. crafting a chair vs. selling a chair.
- Select the input text:
  - From a textual source
  - From interviews with (domain) experts

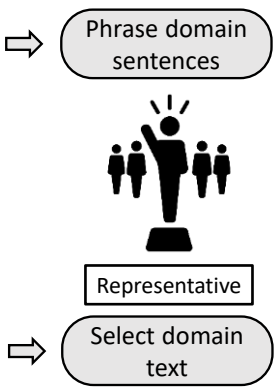


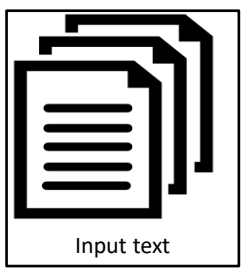
9

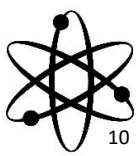


Scope

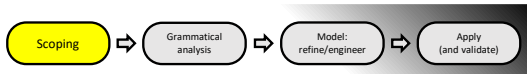
Knowledge source



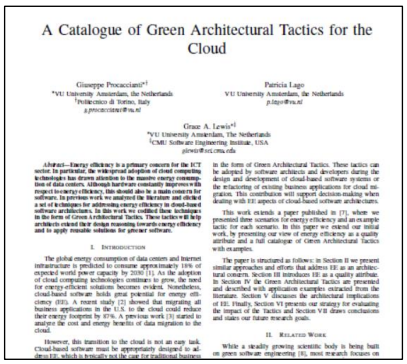




10



Scope from document: an example


A Catalogue of Green Architectural Tactics for the Cloud

Giuseppe Procaccianti[1,2]
[1]VU University Amsterdam, the Netherlands
[2]Politecnico di Torino, Italy
g.procaccianti@vu.nl

Patricia Lago
VU University Amsterdam, the Netherlands
p.lago@vu.nl

Grace A. Lewis[3,1]
[1]VU University Amsterdam, the Netherlands
[3]SEI Software Engineering Institute, USA
glewis@sei.cmu.edu

Abstract—Energy efficiency is a primary concern for the ICT sector. In particular, the widespread adoption of cloud computing technologies has drawn attention to the massive energy consumption of data centers. Although hardware constantly improves with respect to energy efficiency, this should also be a main concern for software. In previous work we analyzed the literature and elicited a set of techniques for addressing energy efficiency in cloud-based software architectures. In this work we codified these techniques in the form of Green Architectural Tactics. These tactics will help architects extend their design reasoning towards energy efficiency and to apply reusable solutions for greener software.

I. Introduction

The global energy consumption of data centers and Internet infrastructure is predicted to consume approximately 10% of expected world power capacity by 2030 [1]. As the adoption of cloud computing technologies continues to grow, the need for energy-efficient solutions becomes evident. Nonetheless, cloud-based software holds great potential for energy efficiency (EE). A recent study [2] showed that migrating all business applications in the U.S. to the cloud could reduce their energy footprint by 87%. A previous work [3] started to analyze the cost and energy benefits of data migration to the cloud.

However, this transition to the cloud is not an easy task. Cloud-based software must be appropriately designed to address EE, which is typically not the case for traditional business

in the form of Green Architectural Tactics. These tactics can be adopted by software architects and developers during the design and development of cloud-based software systems or the refactoring of existing business applications for cloud migration. This contribution will support decision-making when dealing with EE aspects of cloud-based software architectures.

This work extends a paper published in [7], where we presented three scenarios for energy efficiency and an example tactic for each scenario. In this paper we extend our initial work, by presenting our view of energy efficiency as a quality attribute and a full catalogue of Green Architectural Tactics with examples.

The paper is structured as follows: in Section II we present similar approaches and efforts that address EE as an architectural concern. Section III introduces EE as a quality attribute. In Section IV the Green Architectural Tactics are presented and described with application examples extracted from the literature. Section V discusses the architectural implications of EE. Finally, Section VI presents our strategy for evaluating the impact of the Tactics and Section VII draws conclusions and states our future research goals.

II. Related Work

While a steadily growing scientific body is being built on green software engineering [8], most research focuses on

An existing article

Scoping

Together with representative e.g., author


III. Energy Efficiency as a Quality Attribute

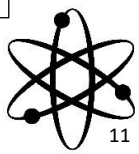
According to Bass et al. [15], EE is to be regarded as a "system" quality attribute because it is the result of an indirect action of software. However, Bass et al. also argue that the line between "software" and "system" quality attributes is very thin. In the end, even if energy is ultimately consumed by hardware, it is software that determines hardware behavior. In order to provide a clear representation of EE as a quality attribute, we follow the approach adopted by Bass et al. [15] and characterize EE through quality attribute scenarios. We grouped our Green Architectural Tactics in three categories and formulated a scenario for each category (see Table I). In all the scenarios, the response measure is energy consumption values. In the following section, we describe the identified scenarios for each category, as well as the elicited Green Architectural Tactics.

IV. Green Architectural Tactics

In previous work [9] we identified a set of recurring design solutions, described in the literature, to achieve EE in cloud-based software architectures. In this work, we codified these solutions as tactics – that is, "design decisions that influence the achievement of a quality attribute response" [15]. Each tactic is described in terms of:

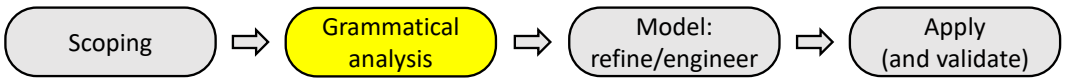
- Motivation: rationale behind the tactic.
- Description: components introduced by the tactic and their roles.
- Constraints: necessary conditions for applying the tactic in an existing software architecture.

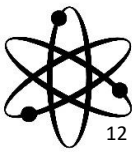
Selected input text



11

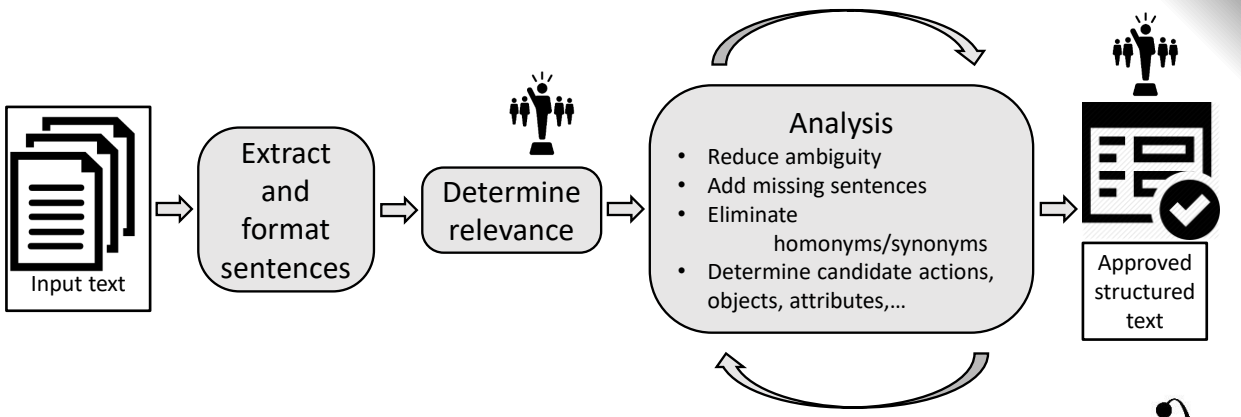
The domain modeling process

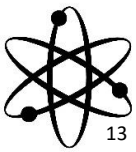




12

Grammatical analysis



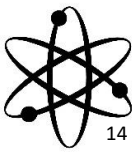


13

13

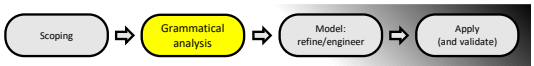
3 sentence formats

- Active sentences: <subject> <predicate> <direct object> <indirect object>*
  - We power a device
  - We install software on a platform
  - We monitor energy consumption
- Static relations: <subject> (has|consists of|contains) <direct object>
  - A device consists of software and hardware
  - A device has a device model
- Nominal predicate: <subject> is <nominal part of the predicate>
  - Energy efficiency is a quality attribute
  - A power meter is a device
  - Wireless communication is a capability



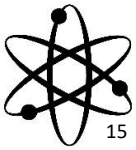
14

14

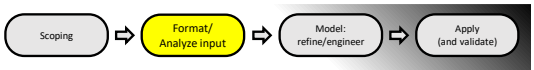


Grammatical analysis

Original sentence	Extracted DM sentences	Questions/remarks
Instrumenting a data center with power metering devices is becoming common practice.	To instrument data center with device. Instrumenting is a practice.	what role does practice play here?
The market is flooded with many different models of power meters with enhanced capabilities (e.g., wireless communications, high sampling frequencies, data analysis features).	To flood market with model. Power meter has model. Power meter has capability. To enhance capability. Wireless communications is a capability. High sampling frequencies is a capability. Data analysis features is a capability.	What is to flood exactly?

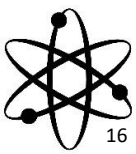


15

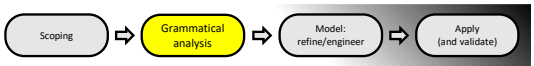


After some iterations

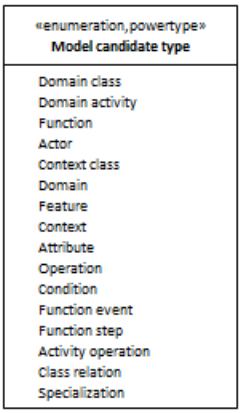
Original sentences	Resulting sentences	Decisions
Instrumenting a data center with power metering devices is becoming common practice.	To instrument data center with device. ~~Instrumenting is a practice.~~	Practice is contextual info.
The market is flooded with many different models of power meters with enhanced capabilities (e.g., wireless communications, high sampling frequencies, data analysis features).	~~To flood market with model.~~ Power meter is a device. Device has device model. Device model has capability. ~~To enhance capability.~~ ~~wireless communications is a capability.~~ ~~high sampling frequencies is a capability.~~ ~~data analysis features is a capability.~~	Purpose is specification of EE tactics, not to market power meters. The design and production of devices is out of scope. So, to enhance is out of scope. The mentioned capabilities are instances/examples.



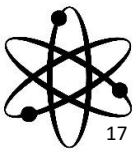
16



Identify candidates

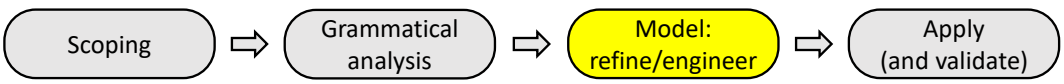


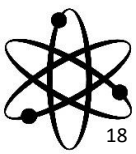
- Grammatical subjects and objects are candidates for domain classes.
- Active verbs are candidates for domain activities.
- and some more
- Some criteria:
  - Instances of Activities and Classes have own identity.
  - Domain objects have an interesting life.
  - Activities are related to classes, i.e., verbs are related to nouns.



17

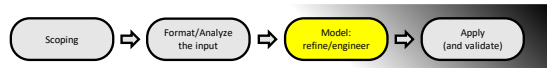
Model engineering





18

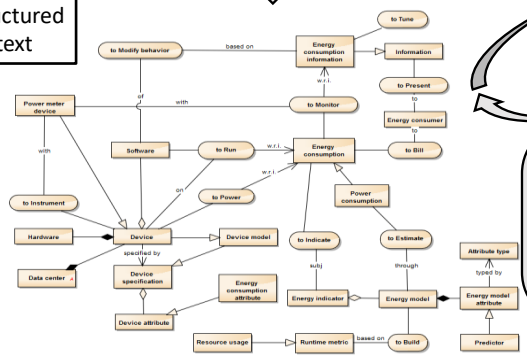
Modeling



Approved structured text

Initial model

- Identify domains
- Put sentences in model



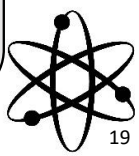
Refine content

- Scope precision
- Detail, e.g., multiplicities
- Identify gaps
- Concept definitions

Expert decides

Engineer

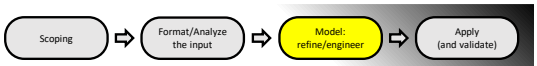
- Filter out derived concepts
- Objectify activities into classes
- Abstract classes to reuse associations
- Check consistency between domain views



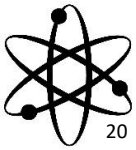
19

19

Guidelines/rules examples

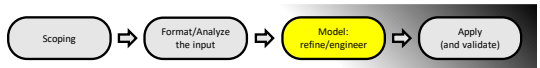


- Introduce a different domain class if and only if:
  - It has different domain actions on it then on another class (you do something different with it)
  - It has different attributes
  - It has an interesting life in the considered scope (it is involved in multiple domain activities). If this is not the case, then it is not a domain class. But more likely a context class.
- A composition of a whole and a part is only interesting if:
  - The part is reused in other compositions.
  - Several parts are instances of the same type (like your left and right eye, or the wheels of a car).
  - There is a need to communicate about the part independently from the whole.
- Only introduce a specialization if:
  - Two or more classes share the same relation to another class or action.
  - Two ore more classes share attribute definitions. (In this case they are probably also sharing a relation/activity).
  - A clear case for the above in the future of the model. (Yes, this is a vaguer criterion).



20

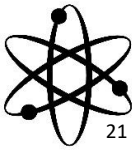
20



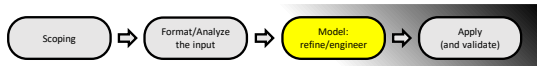
Consistency rule examples

- Each association between classes (including aggregations/compositions):
  - Must be instantiated/created in at least one activity,
  - Or must be part of a composition and the composition has an instantiating activity,
  - Or is created outside the domain.

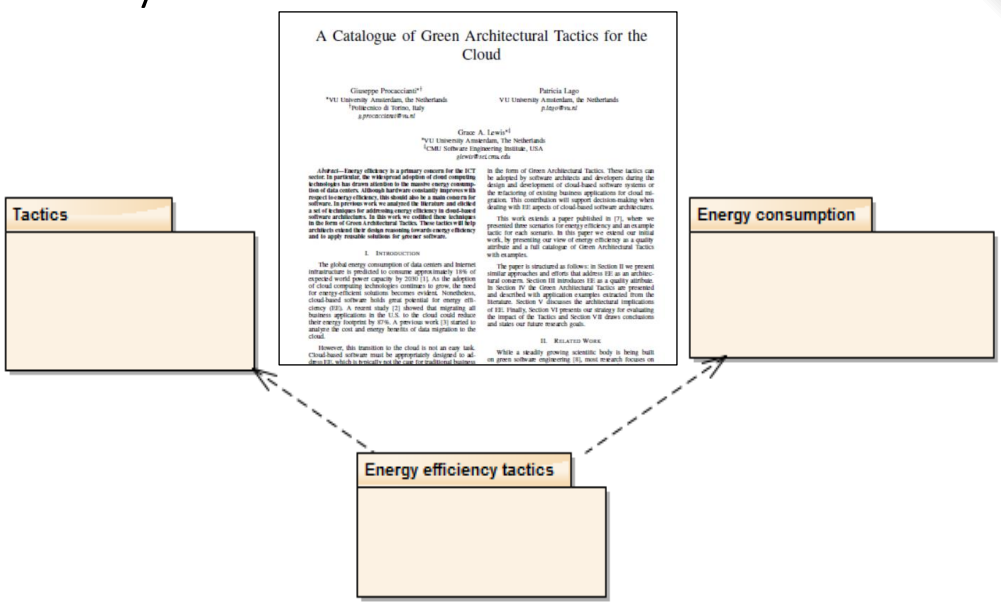
- All attributes must get a value in at least one activity.
- All domain objects must have at least two actions in which they can participate -> All domain classes have at least two activities.



21

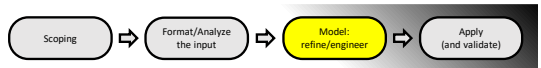


Identify domains

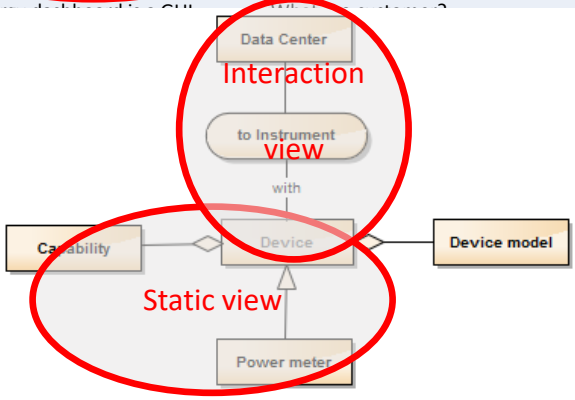


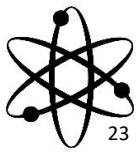
22

From sentences to model

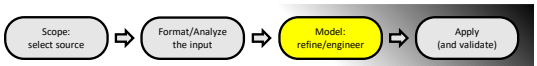


Original sentences	Grammatical extraction	Question/remark	Model status
Instrumenting a data center with power metering devices is becoming common practice. The market is flooded with many different models of power meters with enhanced capabilities (e.g., wireless communications, high sampling frequencies, data analysis features).	To instrument data center with device Power meter is a device Device has device model Device has capability.		Processed
Finally, a GUI component called an Energy Dashboard provides graphical representations of energy information along with useful reporting for both cloud service providers and customers.	Ene com Ene rep info Serv Cus		Postponed to making the model of the metering tactic

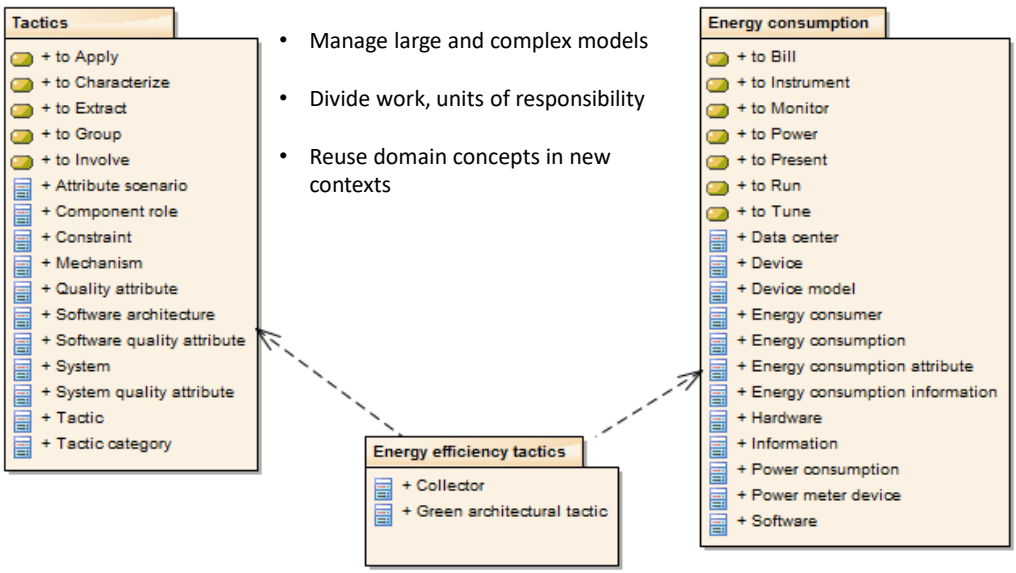




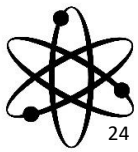
23



Put sentences in domain packages

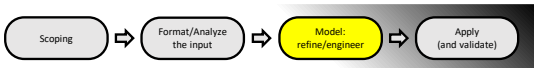


- Manage large and complex models
- Divide work, units of responsibility
- Reuse domain concepts in new contexts



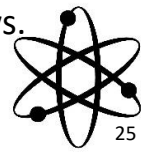
24

Iterating over domain views



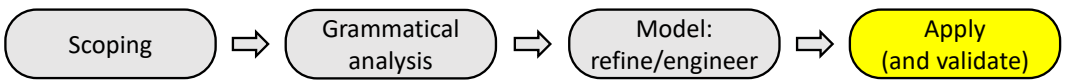
- Interaction view: interactions between classes in terms of domain activities.
- Static view: relates domain classes by associations and generalizations.
- Element definitions view (often just a table, possibly generated).
- Attribute view per domain class and domain activity: the properties of instances of domain activities and domain classes in relation to concepts from the context.
- One or more context models, which contain definitions of objects and operations that are needed for definitions of domain activities and domain classes, but that are not defined by the domain.
- Object lifecycle view per domain class. The possible life of objects of a domain class. (Regular expression, process algebra, activity diagram, or state transition diagram).
- An activity operations view per domain activity to describe logic: pre-/post-conditions, operations.

All viewpoints help in engineering. In practice, use only needed views.

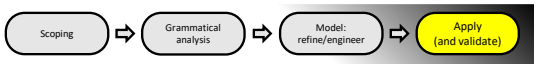


25

The domain modeling process



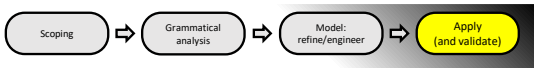
26



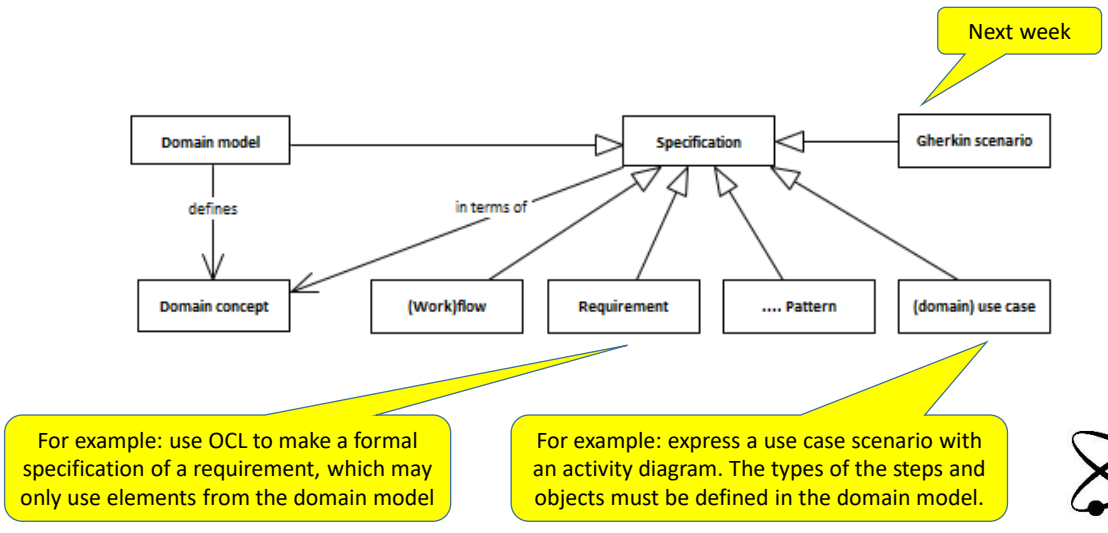
Apply a domain model

- To define consistent specifications, e.g., requirements, use cases.
- To derive other specifications, e.g., (software) design.

27



Domain-based specifications



28

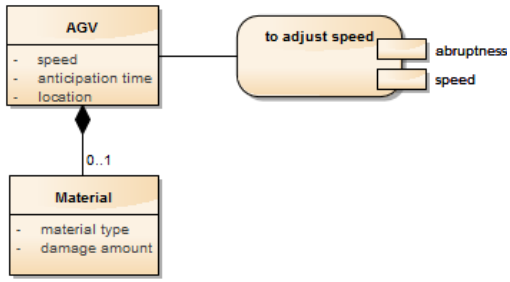
Example of domain-based requirement

If the material on the AGV is glass, then the abruptness of slowing down should not be too big.

Forall actions of the type “to adjust speed” holds: if the AGV (of which the speed is adjusted) is loaded with glass material, and the set speed is lower than the current speed, then the abruptness of the adjust speed is not too big.

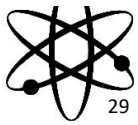
OCL:

Forall (AS: to adjust speed |
AS.AGV.material.material type = “glass”
and AS.AGV.speed > AS.speed
implies AS.abruptness < “too big”)

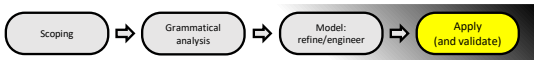


What is “abruptness”?

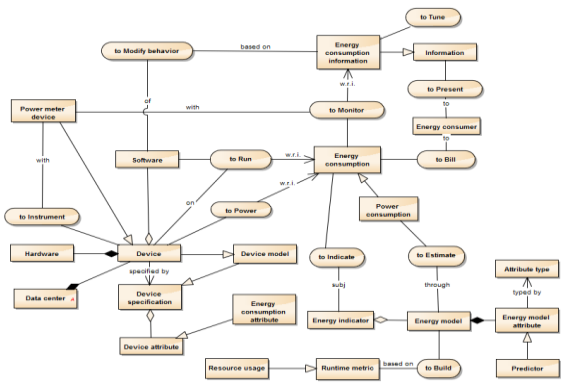
Apparently, the domain model was not precise enough.

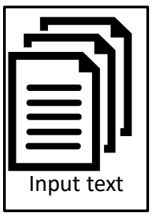


29



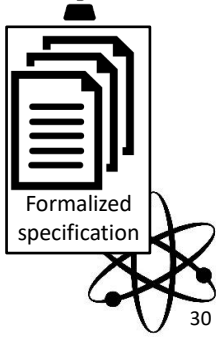
Write domain-based texts/specs





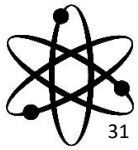
Rewrite original text

Express new specifications

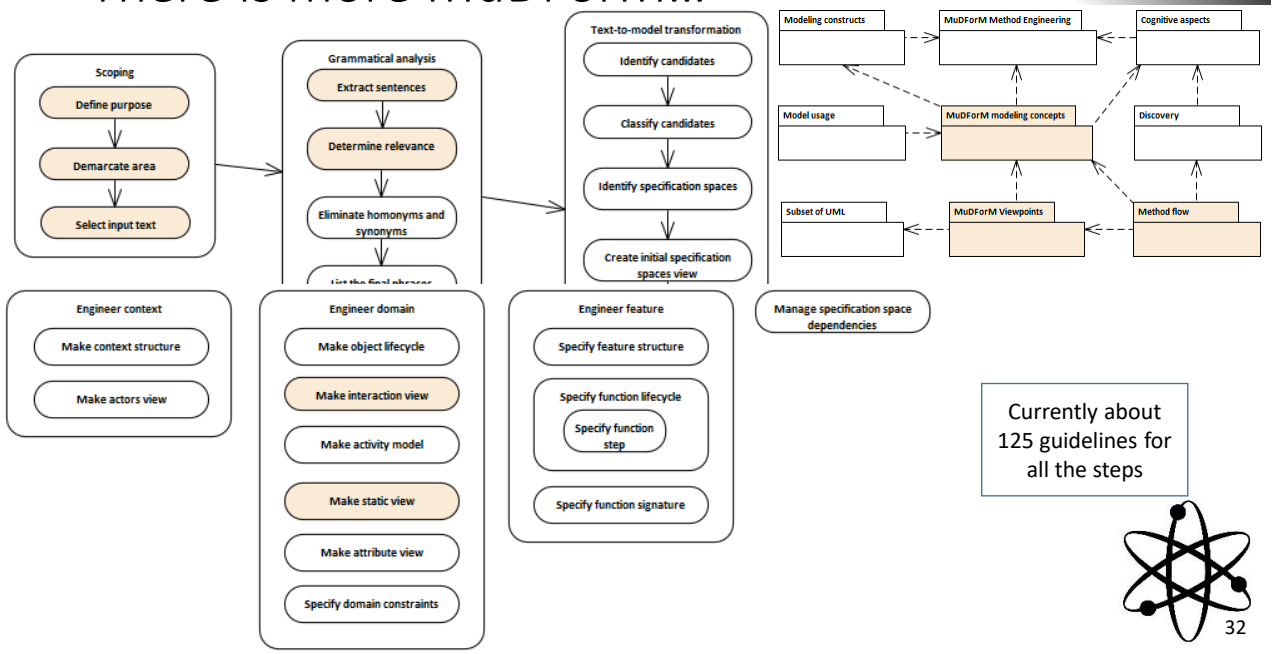


30

In conclusion…



There is more MuDForM…

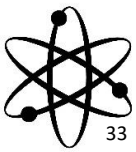


Currently about 125 guidelines for all the steps

Exploit domain models

- As a language:
  - Common/shared by domain experts.
  - Transferable to developers/designers.
  - Also applicable to solution domains or quality attribute domains.
  - Separate analysis from design discussions.
  - For learning the domain.

- Basis for Specifications:
  - Completeness and consistency.
  - Basis for a DSL.
  - Add precision to requirements and functional specifications.



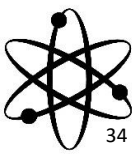
33

Some issues with the current practice are (partially) solved

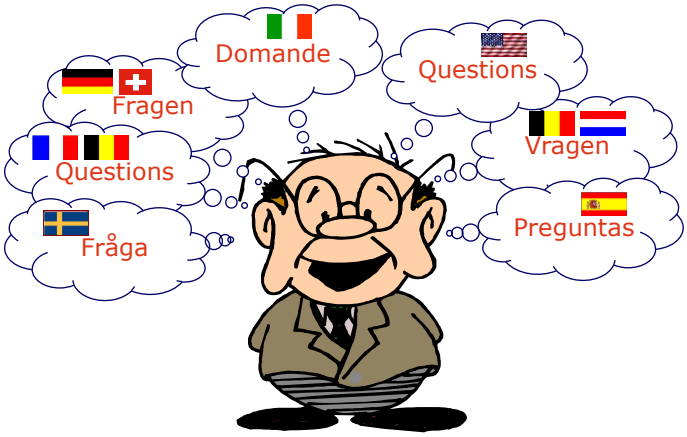
because MuDForM

- is a method:
  - detailed steps and guidelines to guide the modelling activity.
  - Underlying metamodel
- sees behavior as first class citizen, and is not only data/state oriented.
- explicitly links to human communication (in natural language).
- separates application/feature knowledge from domain knowledge.
- offers support making specifications with the domain model.
- explicitly supports multi-domain and multi-feature aspects.

(Yes, not all these points were addressed equally in this presentation.)



34



robert.deckers@AtomFreeIT.com
+31 6 46882428

35

35