



MBE & AI: WHAT ABOUT SYNERGIES?

Sébastien Gérard (*Director of Research at CEA List, sebastien.gerard@cea.fr*)

Département d'Ingénierie des Logiciels et des Systèmes

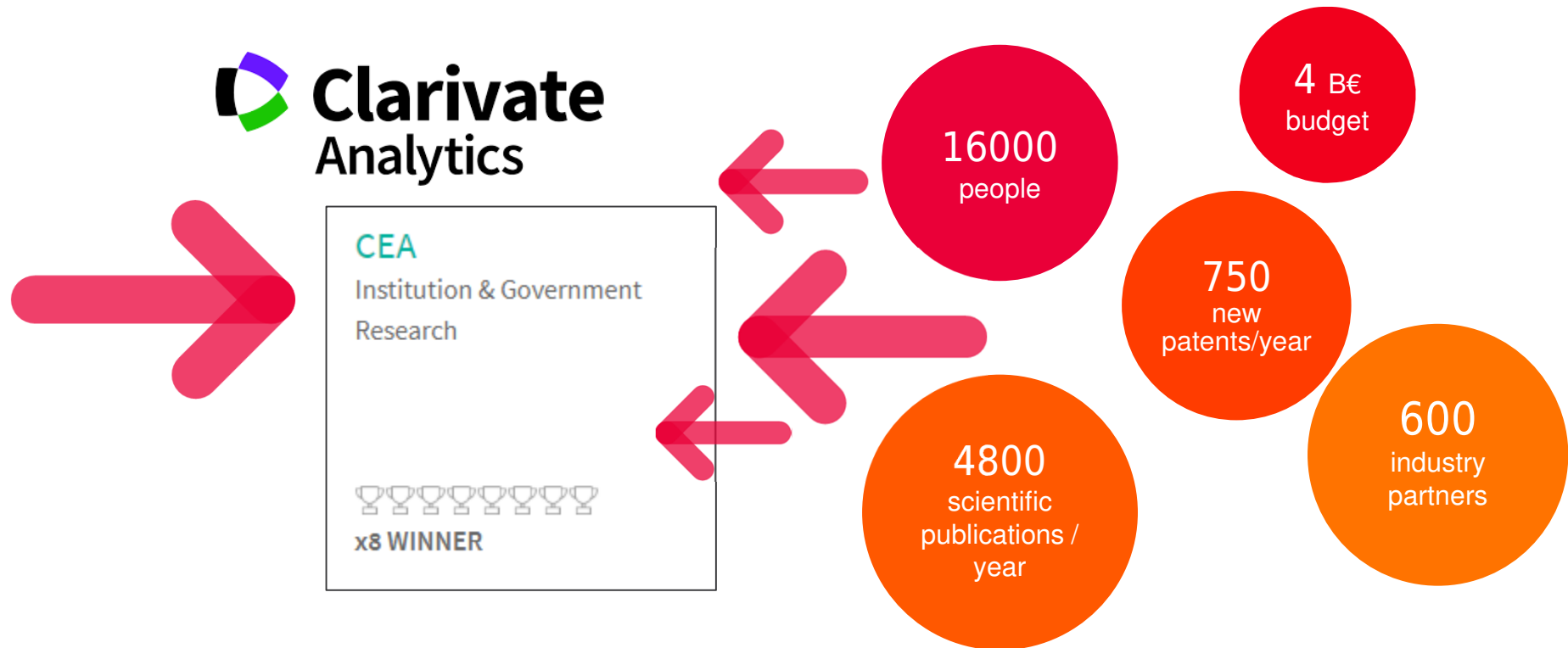
Software and System Engineering Department





CEA, a French public research organisation

“Top 25 Global Innovators – Government” ranking of Reuter:
→ in 2019, CEA is ranked at the 3rd place worldwide in terms of innovative public research organisation.

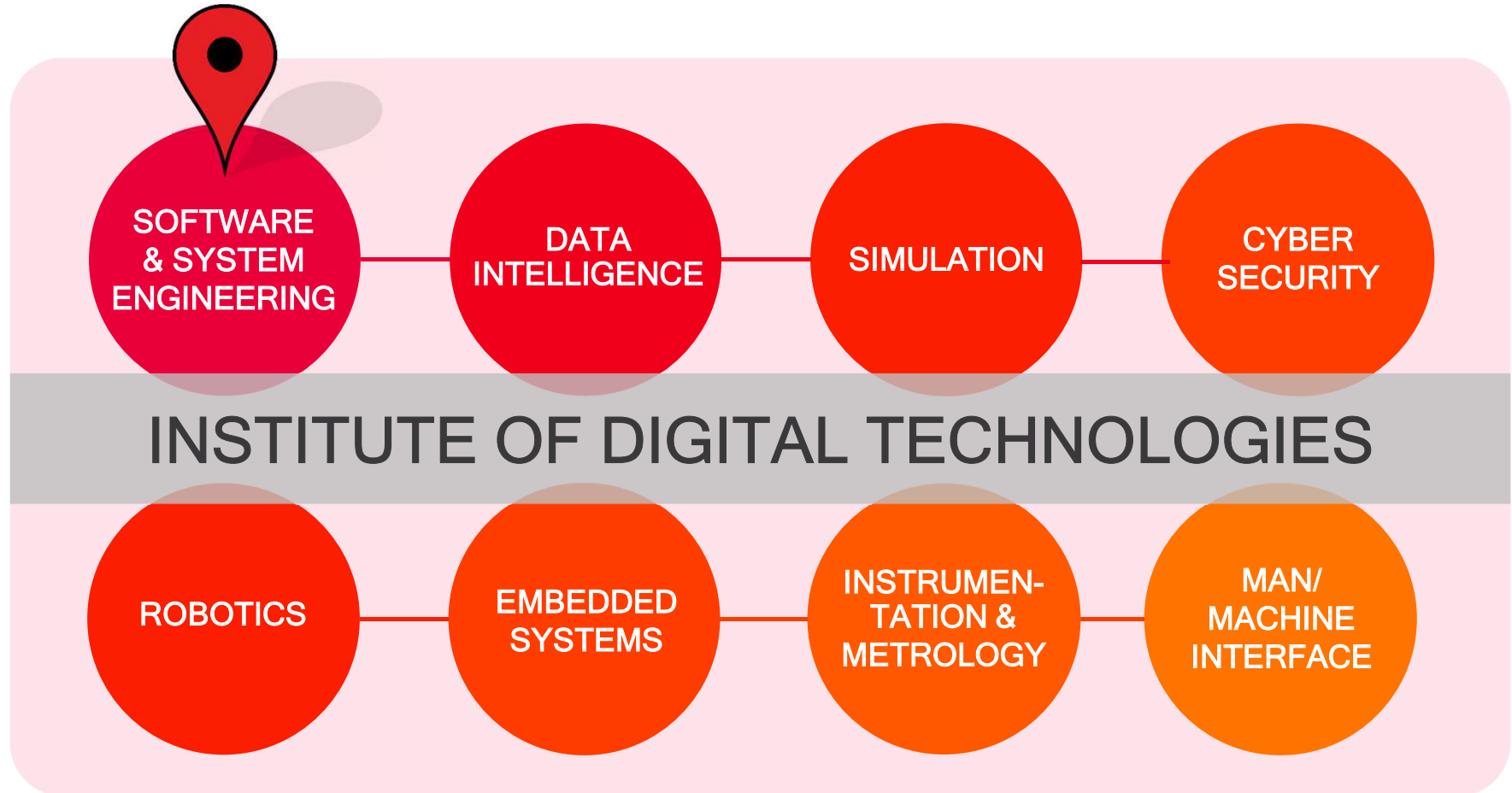


The missions of CEA





Expertise areas of the CEA List institute

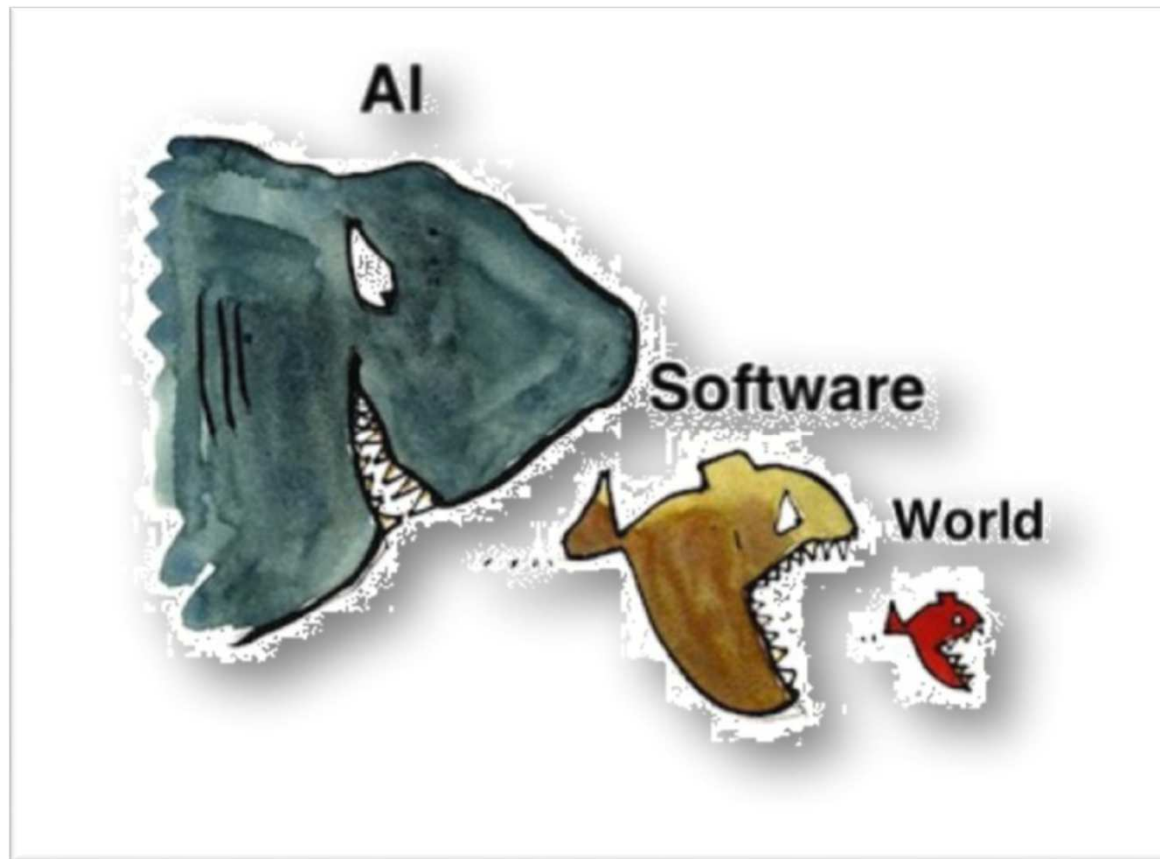


Before going further...



**This presentation will
not be a tutorial on AI...**

**... fortunately, because
I am not an expert in AI!**



“Software is eating the world, but AI is going to eat software”
Jensen Huang, Nvidia CEO (2017)

(Credit to Jordi Cabot for the slide)

More and more publications and communications around how AI for SE....

Computer Languages, Systems & Structures 50 (2017) 159–176

Contents lists available at ScienceDirect

ELSEVIER

Computer Languages, Systems & Structures

Journal homepage: www.elsevier.com/locate/colss

User-story driven development of multi-agile process fragment for agile methods

Yves Wautelet^{a,*}, Samed Heng^b, Soreangy Kiv^b, Mani

^a Centre for Information Management, KU Leuven, Wilmansberg 26, Brussel 1000, Belgium
^b LIAISON/CEIAS, Université catholique de Louvain, Place des Croix, 1, 1348 Louvain-la-Neuve, Belgium

Science of Computer Programming 191 (2020) 102416

Contents lists available at ScienceDirect

ELSEVIER

Science of Computer Programming

www.elsevier.com/locate/scp

Automated reasoning framework for traceability management of system of systems

Bedir Tekinerdogan^{a,*}, Ferhat Erata^{b,c}

^a Information Technology, Wageningen University, the Netherlands
^b Department of Computer Science, Yale University, CT, USA
^c UNT Information Technologies, Research & Development, Turkey

ARTICLE INFO

Received 17 March 2019
 Received in revised form 25 January 2020
 Accepted 26 January 2020
 Available online 3 February 2020

Keywords:
 System of systems
 Traceability
 Reasoning
 Domain specific language

ABSTRACT

An important aspect in system of systems (SoS) is the realization of the SoS in different systems that work together. Identifying and locating these capabilities in order to orchestrate the overall activities and hereby to achieve the overall SoS. System elements and capabilities in SoS however, are rarely stable at evolve in different ways and different times in accordance with the changing requirements. To manage the SoS and cope with its evolution it is necessary that the dependent capabilities and the system elements can be easily traced. Several approaches have been proposed to model traceability and reason about these by extending a pre-set of possible trace links with trace semantics. However, for the context of a traceability model with fixed traceability semantics is limited to consider different and changing scenarios. In this article, we first present the different traceability and traceability analysis approaches within the evolving SoS context to provide the tool support for automated reasoning of traceability of SoS capital system elements. We illustrate and discuss the approach for the application to a SoS.

© 2020 Elsevier B.V. All rights reserved.

1. Introduction

Whereas traditionally systems were addressing a single domain, current systems have to be composed of multiple elements that need to be integrated in a coherent way. To be able to design, analyze, implement and maintain a so-called systems of systems (SoS) [1], a Systems of Systems Engineering (SoSE) approach is required [2, 1]. Tra SoSE has been heavily used in the defense domain but is now also increasingly being applied to non-defense related areas such as air and auto transportation, healthcare, global communication networks, search and rescue system exploration and many other SoS application domains.

One of the key challenges of SoS besides its complexity is the continuous need for evolution whereby new elements and capabilities are deployed, and unnecessary system elements and capabilities will be given up. The evolution of SoS is inherently continuous, whereby adaptations will be made continuously to meet the changing requirements. It is often not local but systemic in that it impacts multiple system elements and capabilities together.

* Corresponding author.
 E-mail address: bedir.tekinerdogan@wur.nl (B. Tekinerdogan).
 https://doi.org/10.1016/j.scp.2020.102416
 0167-6423/© 2020 Elsevier B.V. All rights reserved.

Universal Programmability - How I

Walter F. Tichy, Mathias Landhäuser, Sven J. Tichy | landhaeuser@svn.koernerkit.edu

FOR APPLICATION DEVELOPMENT & DELIVERY PROFESSIONALS

How AI Will Change Software Development

By Diego Lo Gulio
 with Christopher Mines, Amanda LeClair, Rowan
 October 13, 2016 | Updated: November 2, 2016

Everyone should be able to program. Programming in informal, but precise natural languages would enable anyone to program and help eliminate the world-wide software backlog. Highly trained software engineers would still be needed for complex and demanding applications, but not for routine programming tasks.

Programming in natural language is a monumental challenge that will require AI and software researchers to join forces, but, more importantly, to understand and locate these capabilities in order to orchestrate the overall activities and hereby to achieve the overall SoS. System elements and capabilities in SoS however, are rarely stable at evolve in different ways and different times in accordance with the changing requirements. To manage the SoS and cope with its evolution it is necessary that the dependent capabilities and the system elements can be easily traced. Several approaches have been proposed to model traceability and reason about these by extending a pre-set of possible trace links with trace semantics. However, for the context of a traceability model with fixed traceability semantics is limited to consider different and changing scenarios. In this article, we first present the different traceability and traceability analysis approaches within the evolving SoS context to provide the tool support for automated reasoning of traceability of SoS capital system elements. We illustrate and discuss the approach for the application to a SoS.

Automated Validation of Requirement Reviews: A Machine Learning Approach

Maninder Singh
 Department of Computer Science and Engineering
 North Dakota State University
 maninder.singh@ndsu.edu

ABSTRACT

Software development is fault-prone especially during the early phases of requirements and design. Software inspections are commonly used in industry to detect and fix problems in requirements and design artifacts thereby mitigating the fault propagation to later phases where same faults are harder to find and fix. The output of an inspection process is natural language (NL) reviews that report the location and description of faults in software requirements specification documents (RS). The artifact authors must manually read through the reviews and differentiate between true-faults and false-positives before fixing the faults. The time spent in making effective post-inspection decisions (number of true faults and deciding whether to re-inspect) could be spent in doing actual development work. The goal of this research is to automate the validation of inspection reviews, finding common patterns that describe high-quality reviews, identify fault-prone requirements pre-inspection, and interrelated requirements to assist fixation of reported faults post-inspection. To accomplish these goals, this research employs various classification approaches, NL processing with semantic analysis and mining solutions from graph theory to requirement reviews and NL requirements. Initial results w.r.t. validation of inspection reviews have shown that our proposed approaches were able to successfully categorize useful and non-useful reviews.

Keywords: requirement inspections; classification; semantic analysis; topic modeling; high quality requirements; interrelated requirements; part of speech tags

1. INTRODUCTION

Software development involves gathering requirements from various stakeholders (some of whom are non-technical) and producing a natural language (NL) software requirement specification (SRS) document. The SRS document forms basis for downstream software development activities i.e. design, coding, testing. Due to the inherent nature of NL and involvement of multiple stakeholders, requirements are prone to redundancy, inconsistency, and ambiguity.

To verify requirement recorded in SRS, software companies employ peer-reviews to ensure that requirements meet certain quality standards (e.g., correctness, completeness). During peer-reviews, skilled inspectors read through each requirement and report any potential faults, which are then handed back to the requirements author. The requirements author manually read through each reported review to identify useful reviews that report actual (fault) and remove non-useful reviews (false-positives). Additionally, when a true fault is reported in SRS, the author may not manually check other parts of the SRS that are affected (e.g., may contain similar faults) and

Engineering Applications of Artificial Intelligence 26 (2013) 1631–1640

Contents lists available at ScienceDirect

ELSEVIER

Engineering Applications of Artificial Intelligence

Journal homepage: www.elsevier.com/locate/engappai

Linking software testing results with a machine learning approach

Alexandre Rafael Lenz, Aurora Pozo, Silvia Regina Vergilio^{*}

Computer Science Department, Federal University of Paraná (UFPR), Brazil. CP 19.061, CEP: 81531-970, Curitiba, Brazil

ARTICLE INFO

Received 15 March 2012
 Received in revised form 10 January 2013
 Accepted 19 January 2013
 Available online 6 April 2013

Keywords:
 Machine learning
 Software testing
 Test coverage criteria

ABSTRACT

Software testing techniques and criteria are considered complementary since they can reveal different kinds of faults and test distinct aspects of the program. The functional criteria, such as Category Partition, are difficult to be automated and are usually manually applied. Structural and fault-based criteria generally provide measures to evaluate test sets. The existing supporting tools produce a lot of information including: input and produced output, structural coverage, mutation score, faults revealed, etc. However, such information is not linked to functional aspects of the software. In this work, we present an approach based on machine learning techniques to link test results from the application of different testing techniques. The approach groups test data into similar functional clusters. After this, according to the testers' goals, it generates classifiers (rules) that have different uses, including selection and prioritization of test cases. The paper also presents results from experimental evaluations and illustrates such uses.

© 2013 Elsevier Ltd. All rights reserved.

1. Int

The quality fundar to rev test ca fault, achiev criteri apply (contr based test di strate kind o creati

2018 ACM/IEEE 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering

Semi-automatic Generation of Active Ontologies from Web Forms for Intelligent Assistants

Martin Biersch
 Karlsruhe Institute of Technology
 Karlsruhe, Germany
 biersch@kit.edu

Mathias Landhäuser
 thingsTHINKING GmbH
 Karlsruhe, Germany
 mathias@thingsTHINKING.net

Thomas Mayer
 Karlsruhe Institute of Technology
 Karlsruhe, Germany

Intelligent assistants are becoming widespread. A popular method for creating intelligent assistants is modeling the domain (and thus the assistant's capabilities) as Active Ontology. Adding new functionality requires extending the ontology or building new ones, as of today, this process is manual.

We describe an automated method for creating Active Ontologies for arbitrary web forms. Our approach leverages methods from natural language processing and data mining to synthesize the ontologies. Furthermore, our tool generates the code needed to process user input.

We evaluate the generated Active Ontologies in three case studies using web forms from the UIUC Web Integration Repository, namely from the domains airfare, automobile, and book search. First, we examine how much of the generation process can be automated and how well the approach identifies domain concepts and their relations. Second, we test how well the generated Active Ontologies handle end-user input to perform the desired actions. In our evaluation, EASIS automatically generates 65% of the Active Ontology's sensor nodes; the generated ontology for airfare search correctly answers 79% of the queries.

CCS CONCEPTS

Human-centered computing → Natural language interfaces; Computing methodologies → Ontology engineering; Natural language processing.

ACM Reference Format:

Martin Biersch, Mathias Landhäuser, and Thomas Mayer. 2018. Semi-automatic Generation of Active Ontologies from Web Forms for Intelligent Assistants. In *RAISE '18: ACM/IEEE 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, May 27, 2018, Gothenburg, Sweden. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3193404.3193408>

1 INTRODUCTION

Intelligent assistants such as Amazon's Alexa and Apple's Siri are omnipresent. They cover the basic functions of the computers they run on and use external services to answer questions such as "How's the weather going to be in Gothenburg tomorrow?" Yet adding new capabilities to such assistants is a time-consuming manual task.

Apple's Siri uses Active Ontologies (AOs) to process the end-user's request [1]. When building AOs, developers must explicitly model the domain concepts as nodes and their relationships as directed edges in a hierarchical graph. In the weather example above, we'd need (at least) the following nodes: the action (i.e., deliver the weather report), the place, and the date for the forecast. Specialized leaf nodes, so-called sensor nodes, are responsible to extract relevant information from the end-user's input, e.g., the place node would extract city names. If a sensor node detects relevant information in the input, it sends a message to its parent node. Inner nodes, in contrast to sensor nodes, receive messages from their children and forward them to their parents, e.g., the date node could have a child that interprets words such as "tomorrow" and another child that detects dates such as "May 27, 2018", in doing so, inner nodes can refine, combine, or ignore the information that they receive. The root node models a distinct operation or function of the assistant and calls, for example, a web service. Because of this design, bottom-up input processing is efficient and developers can understand the domain model easily. The downside of this approach is that every time the capabilities of the intelligent assistant must change (e.g., supporting additional options), developers must explicitly model the operation and their parameters. This is a time-consuming task and limits the applicability of AO-based assistants.

EASIS is an AO server and AO building framework for form-based services. Many service providers that target end-user provide HTML forms for accessing them but do not provide standardized web services that could be invoked in the root node. EASIS simplifies building AOs for such forms by guiding the developer through the creation process. First, EASIS crawls the form for HTML forms and categorizes them (e.g., an airfare, automobile, or book search services). In previous work, we explored how clustering techniques can be adapted to this task [3]. This paper concentrates on the generation of an AO for a specific service category. Given minimal web forms providing the same service, EASIS generates an unified AO that captures all characteristics of the different forms (e.g., similar forms from different airlines). In addition, it generates the necessary sensor nodes. If information for the fully automatic AO generation is missing, the developer is asked. Last but not least, it generates a service directory that is used in production to actually call the external services and collect their responses.

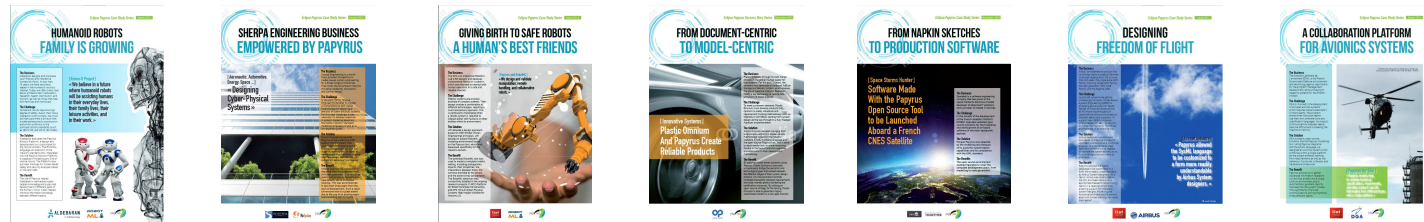
Section 2 explains AOs and the EASIS Active Server and Section 3 reviews related work. The following sections describe how we automatically derive AOs from a collection of web forms and how well these AOs can respond to actual end-user input. The final section presents our conclusions and discusses future work.

AI can contribute all along the software development life cycle e.g.,

- **Software requirements**
 - NLP + Ontology used to generation of conceptual models from text-based specification:
 - e.g., class-based, use-case, statemachine, or even interaction.
 - Machine learning:
 - Requirement review, glossary extraction, requirements classifications & prioritizations, etc.
- **Software design**
 - Bots (Sw agents) to collaboratively build conceptual models,
 - Machine learning for classifying web images as UML static diagrams,
 - Recommender that suggest design solutions to the deigners.
- **Software construction**
 - Machine learning for code generation,
 - NLP for reverse engineering.
- **Etc.**

MBE, an enabler of the digital transition for system & software engineering

From documentware... ...to modelware



Slow effective adoption of MBE due to a variety of social and technical factors, but **the tools are often blamed!**

What about cognifying MBE?



Cognification: “the process of making objects or systems smarter and smarter by connecting, integrating sensors and building software/artificial intelligence into them.”



Cognify MBE to improve drastically its benefits and hence reduce the barriers for its adoption.



Modelia,
a joint R&D partnership between



“Bringing artificial intelligence to the modeling world and reversely.”

Jordi Cabot

ICREA Research Profesor,
UOC / SOM lab



Sébastien Gérard

CEA Research Director,
List

Three ongoing projects:



Loli Burgueño, post-doc
(2019-2020) on
“**Uncertainty & AI-4-MT**”.

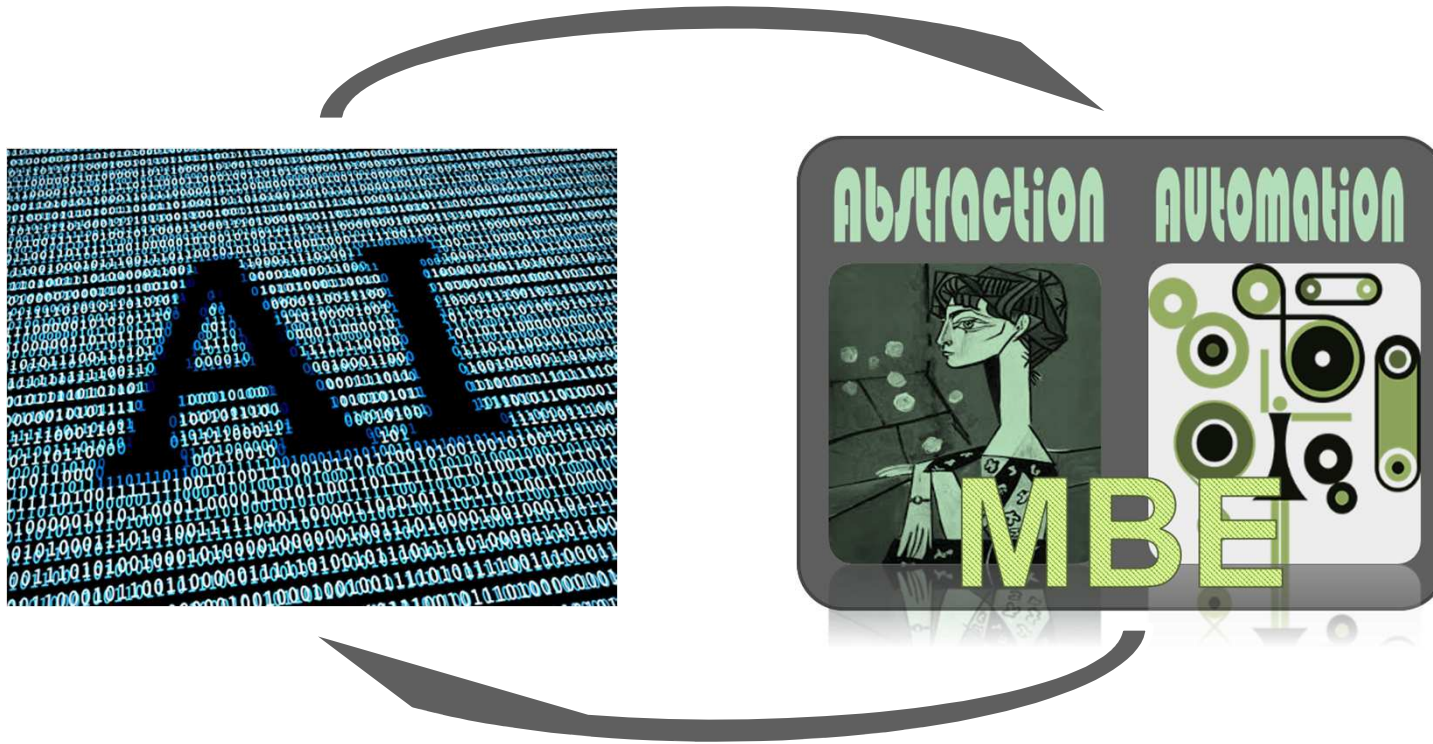


Maxime Savary-Leblanc,
Phd student (2019-2022) on
“**Modeling bots**”.

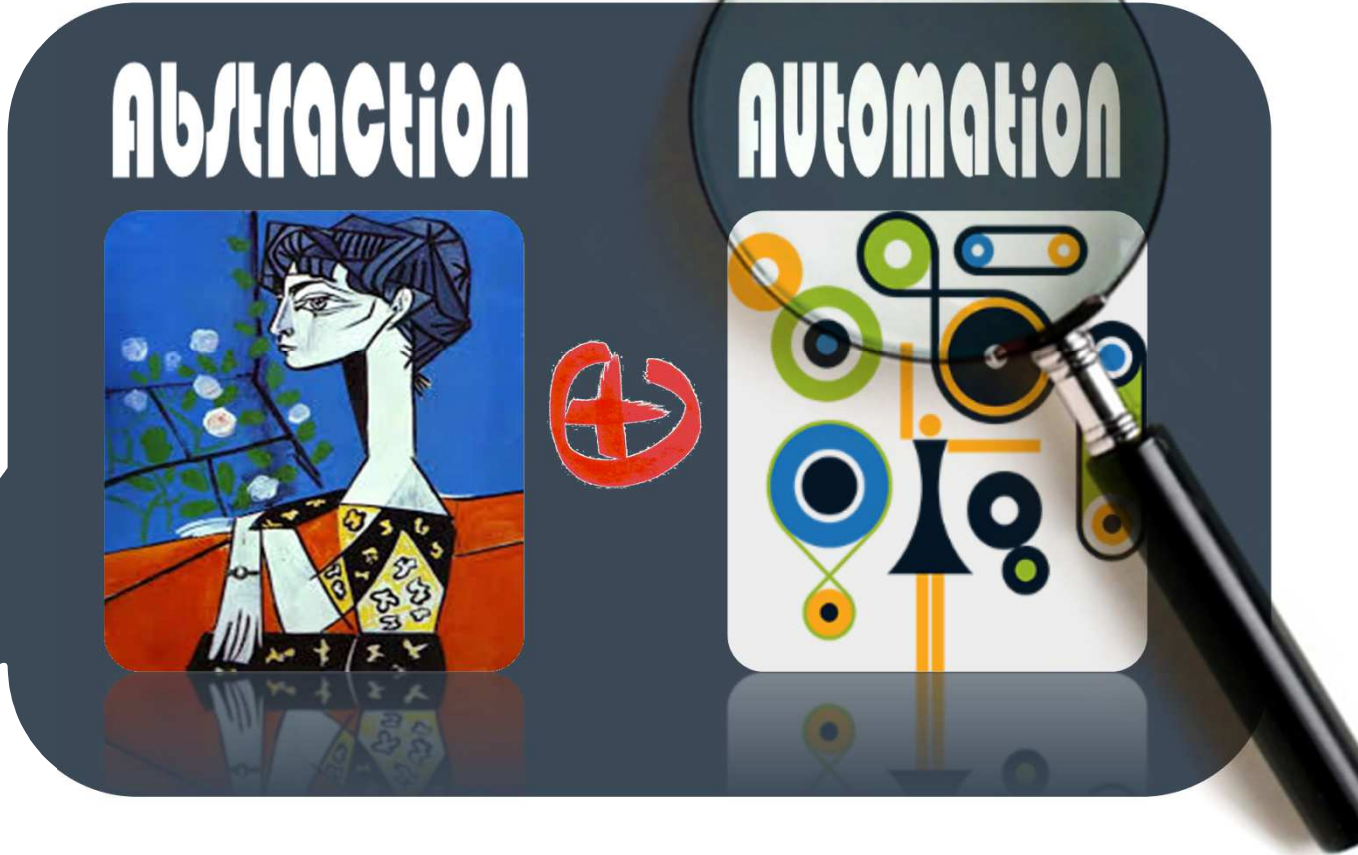


Takwa Kochbati, Phd student
(2019-2022) on “**From Text to
Conceptual Models**”.

AI for MBE, and reversely?



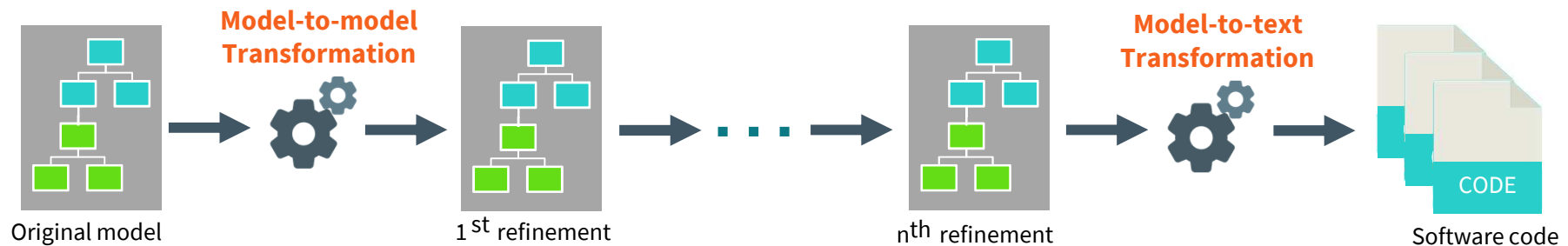
The two main pillars of MBE



B. Selic, "Model-driven development: its essence and opportunities," in Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06), 24-26 April 2006, Gyeongju, Korea.

What is model transformations (MT)

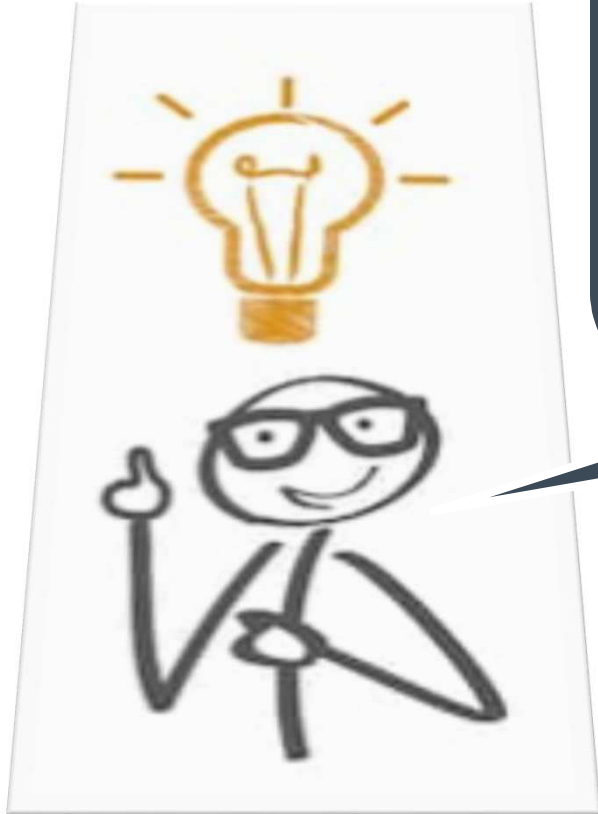
- Model transformations define the mappings between models at a metamodel level



- For people who are not familiar with the modeling paradigm, or with software development and coding in general:
 - Requires learning a new language (the MT Language),
 - Time consuming,
 - Error prone and not easy to debug and maintain.



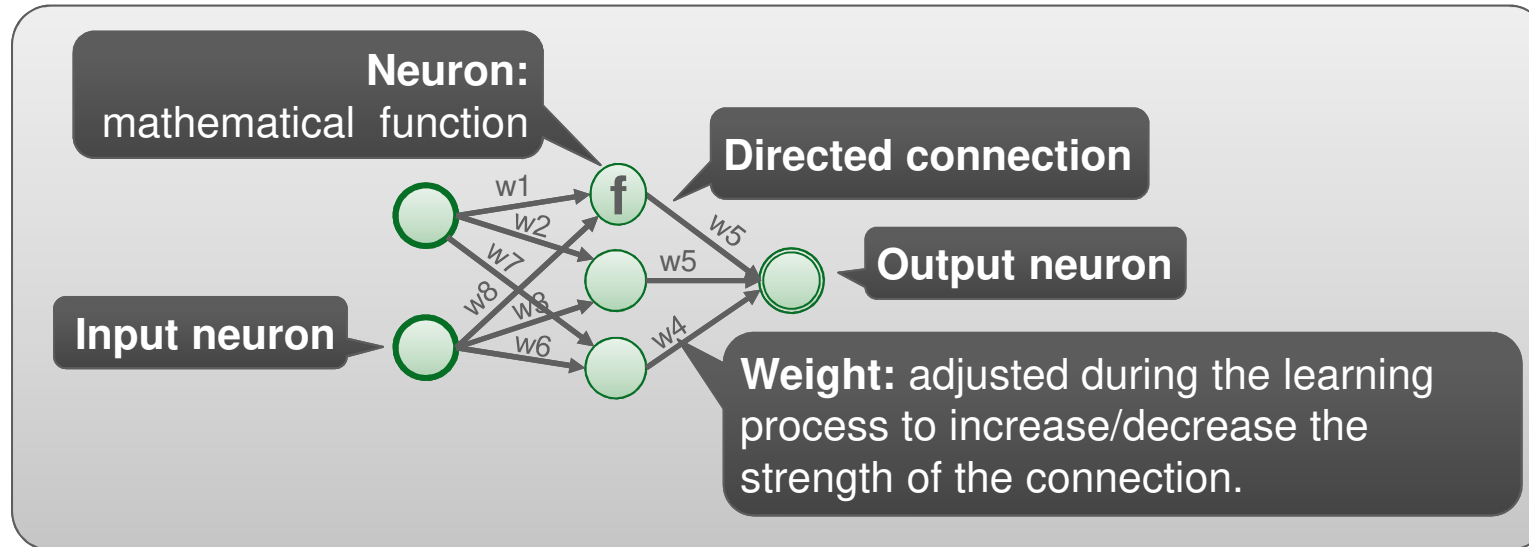
Model transformation ~ Language translation:
 → From SMT / RBMT to ANN approaches.



Artificial Neural Network
to derive model
transformations from
sets of input/output
models.

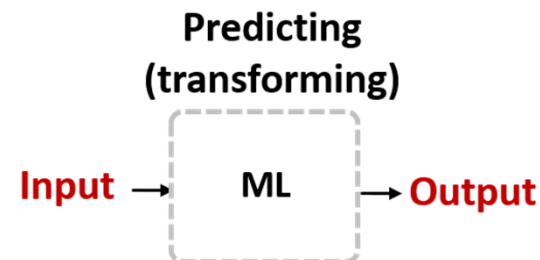
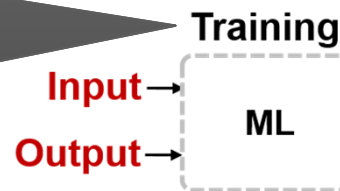
Slide bar – Artificial Neural Network (ANN)

- ANN = directed graph structure of neurons



- Supervised learning process:

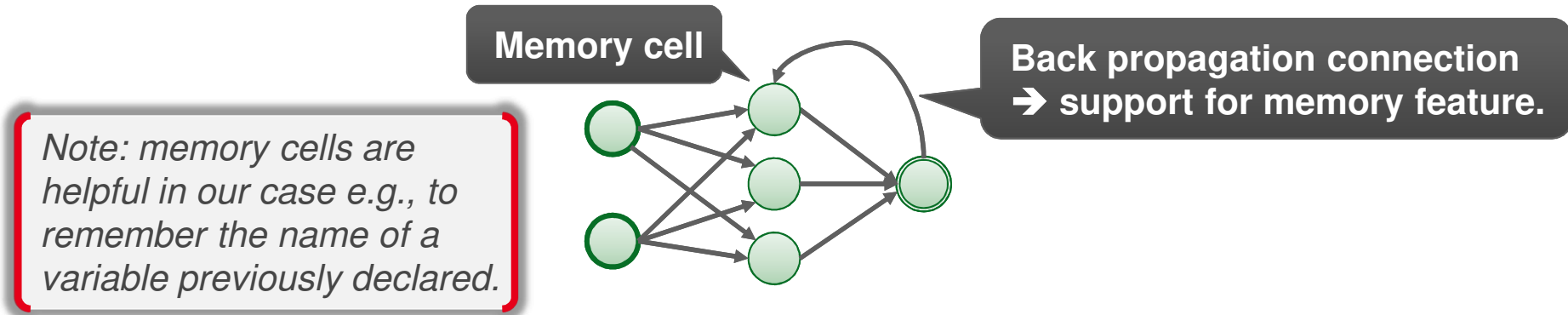
Based on Input-output pairs: 3 datasets for training, validation and testing.



Which ANN architecture to choose?

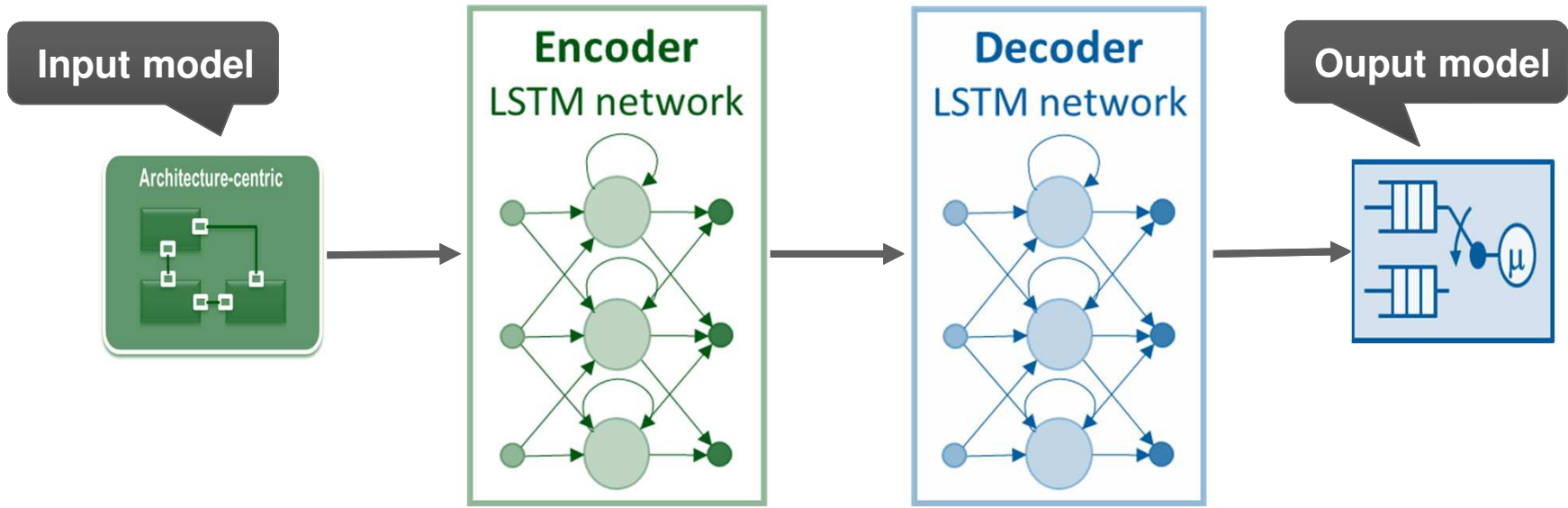


- ANN → Recurrent Neural Networks (RNN)



- RNN → Long Short-Term Memory (LSTM)
 - Longer “memory”,
 - Can remember their context.
- LSTM + Encoder-Decoder architecture
 - Proven to be the most effective architecture of ANN for dealing with translation problems.

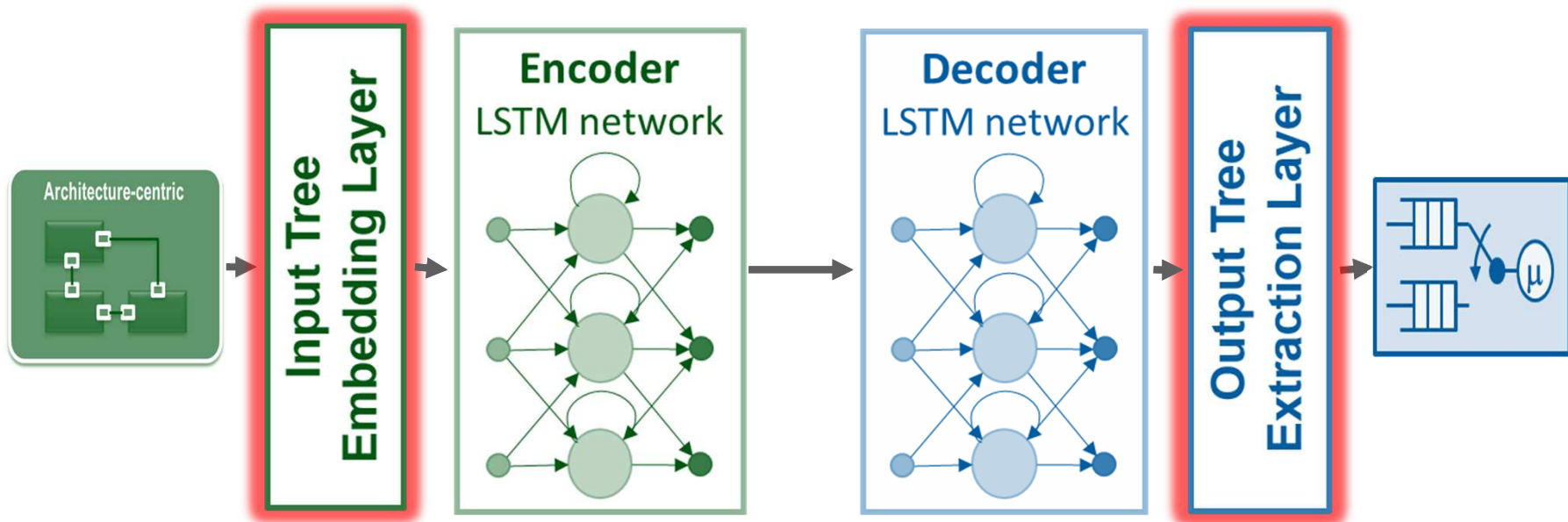
Encoder-decoder architecture + Long short-term memory neural networks



But a model is not a sequence of words...

From Sequence-to-Sequence to Tree-to-Tree!

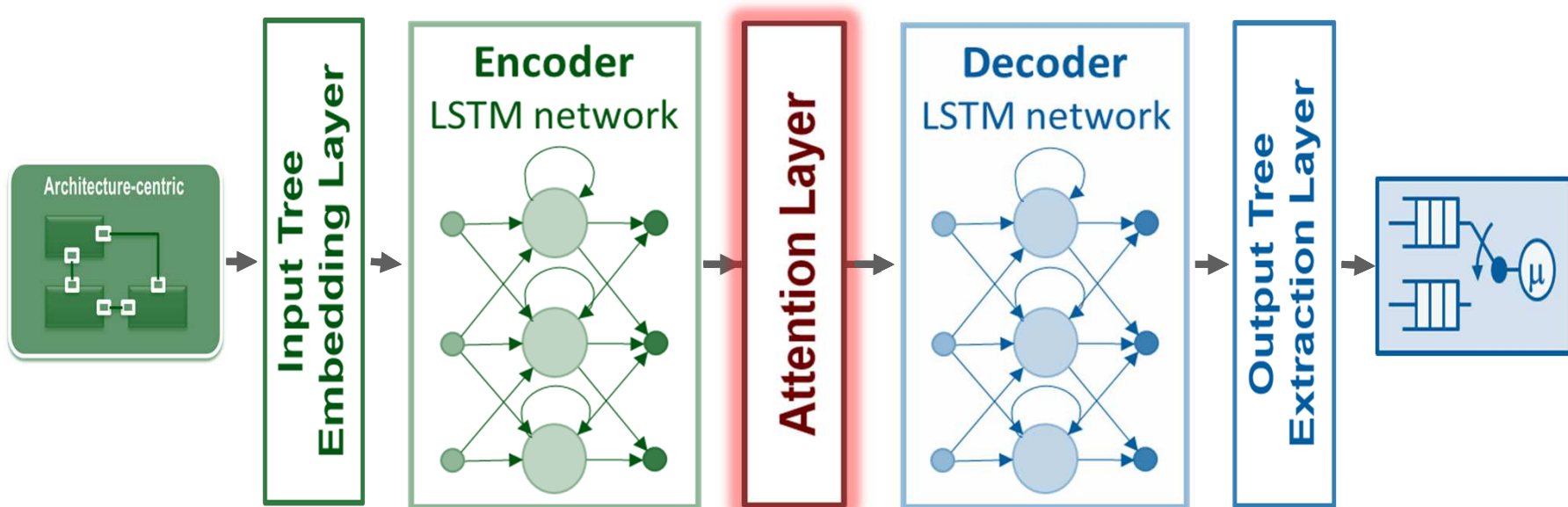
→ 2 additional layers to embed the input tree to a numeric vector and reversely.



And with a bit of attention, it is even better...

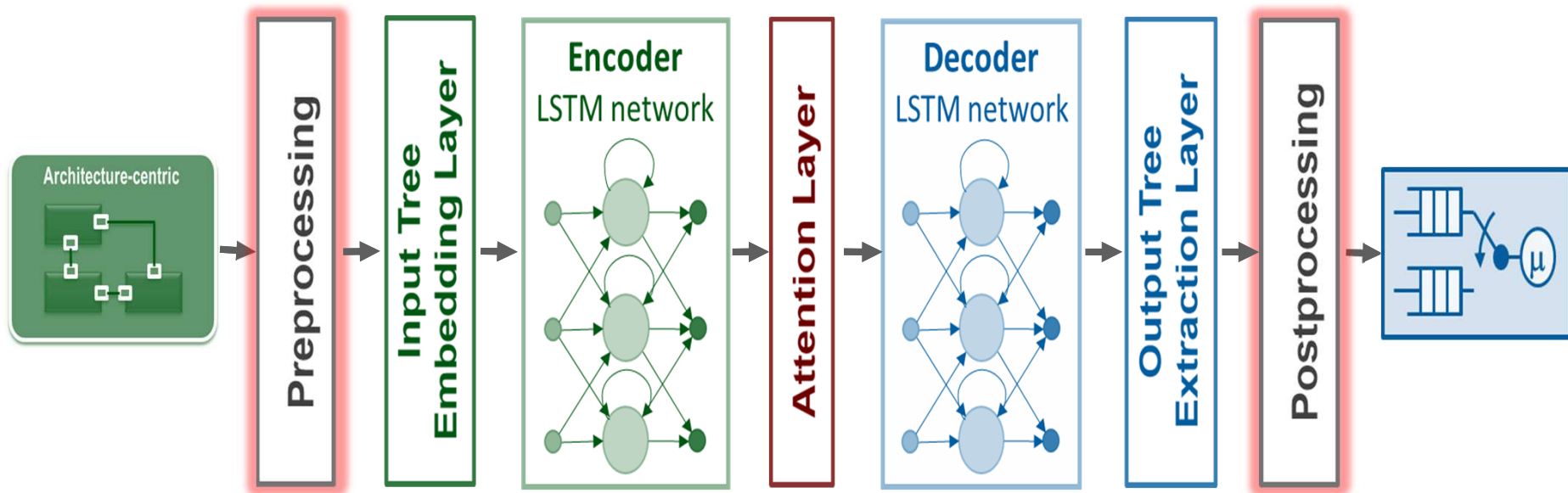
- **Attention mechanism**

- To pay more attention (remember better) to specific parts,
- It automatically detects to which parts are more important.



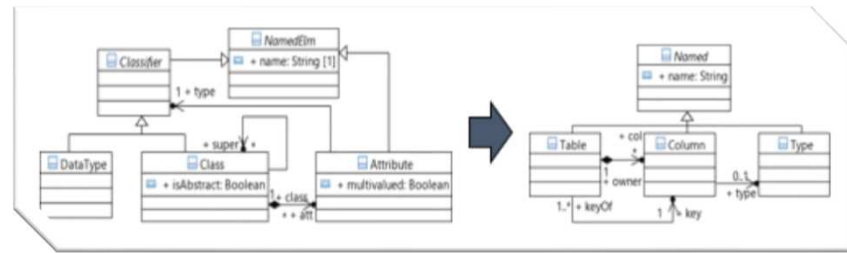
And finally...

- **A bit of pre- and post-processing:**
 - To represent models as trees and conversely,
 - To eliminate symmetries in input models using a canonical form,
 - And to rename variables to avoid the “dictionary problem”.
- **And some configuration of parameters:**
 - Layers#, neurons#, initial weights, learning rates, etc.
 - ➔ Values are results of experiences (black-art of ML...)

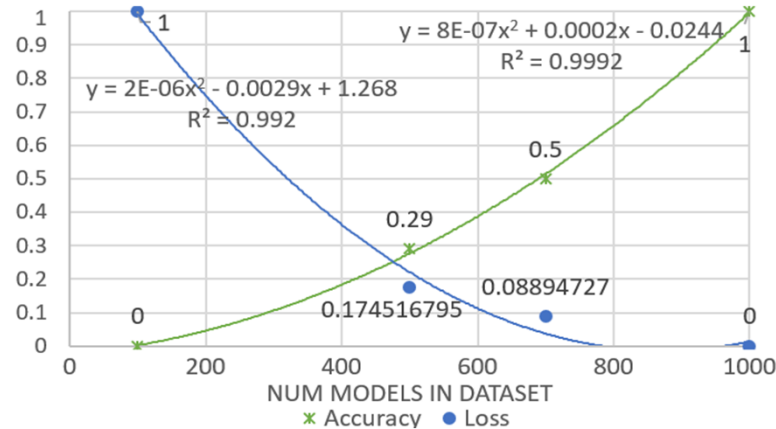


About the results...

(Use case: from Class to Relational models)

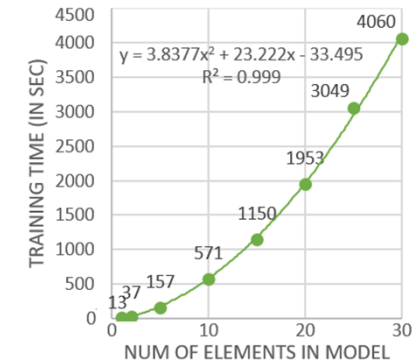
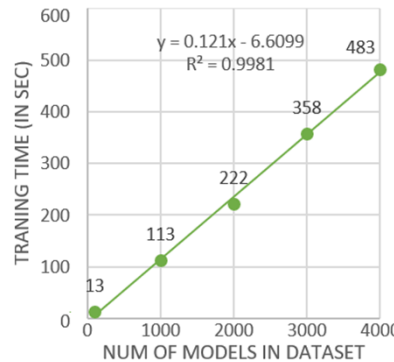


- **Correctness:** measured through the *accuracy* and *loss of the ANN*

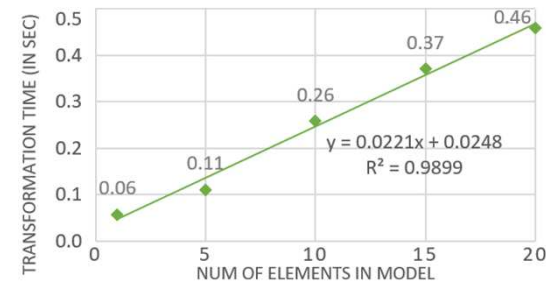


- **Performance**

- How long is the training phase?



- What is the performance of the ML-based MT?



ML-based approach for MT is feasible but obviously there are a number of open challenges to be solved before it can actually be used in practice.

Open Challenges



Size of the training dataset

- 💡 Model mutation procedures or GAN / Transfer learning / pre-trained networks for typical MT scenarios?



Diversity in the training set

- 💡 Coverage metrics for the I/O metamodels / graph kernel techniques?



Generalization problem

- 💡 Transfer learning / common sense ontologies?



Social acceptance & trust

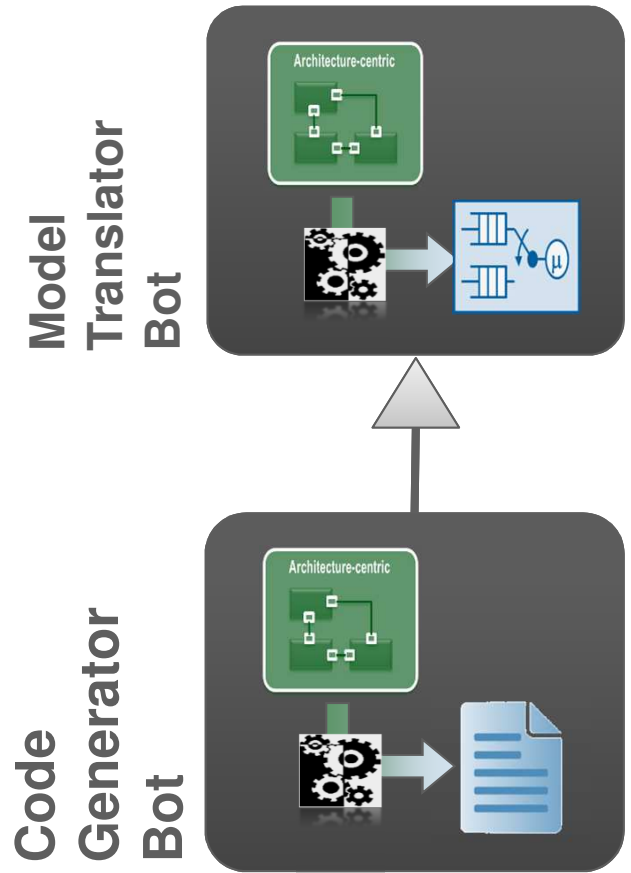
- 💡 Explainable AI, certified AI... ?



Computational limitations of ANNs

- 💡 Expect new results from AI domain...

A new idea...



Open challenges:

- Size of the training dataset,
- Diversity in the training set.

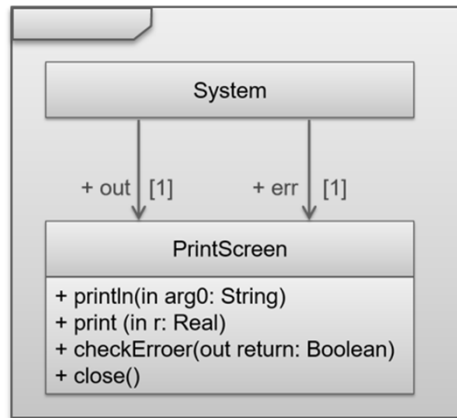


A lot of code available in open-source projects could be reversed into models.

Outlines of the second experience

- **Scope**

UML Class-based Model



Java code

```

public class System {
    private PrintStream out;
    private PrintStream err;

    public PrintStream getOut() {
        return out;
    }
    public void setOut(PrintStream out){
        this.out = out;
    }
    public PrintStream getErr() {
        return err;
    }
    public void setErr(PrintStream err){
        this.err = err;
    }
}
    
```

```

public class PrintStream {
    public void println(String arg0){
    }
    public void print(double d){
    }
    public boolean checkError(){
    }
    public void close() {
    }
}
    
```

- **Main requirements for our code-generator bot**

- Auto-inference of mappings between the structural parts of both source and target,
- Respect coding standards (e.g., imposed by company or community) implicitly available in the dataset used to train the ANN.

About the coding standard rules, the bot needs to learn...

De facto standard rules:

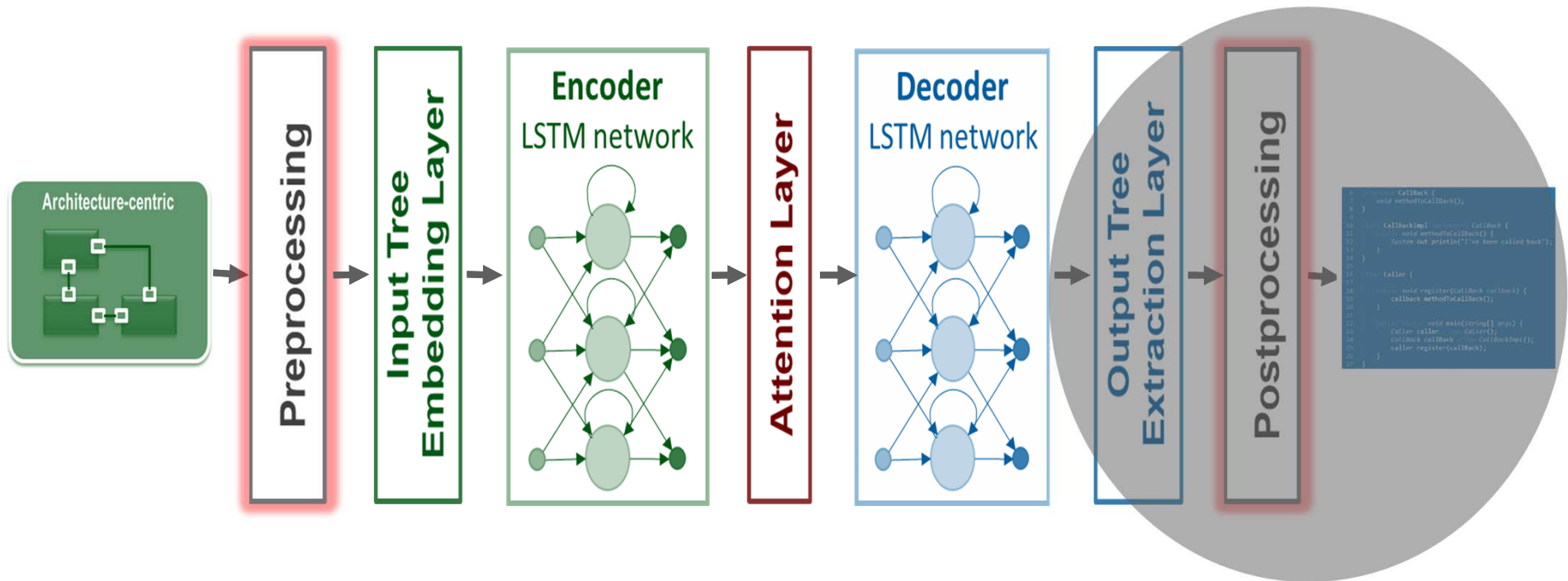
- UML classes into Java classes,
- UML attributes into Java attributes,
- UML associations into Java attributes.
- Etc.



Ad hoc rules (company specific):

- Primitive data type conversions (e.g., a UML Real into a Java double or float),
- Visibility management (public UML attributes into Java public attributes or as a private attributes + getter and setter public methods).
- Etc.

Adaptated ANN architecture for code generator



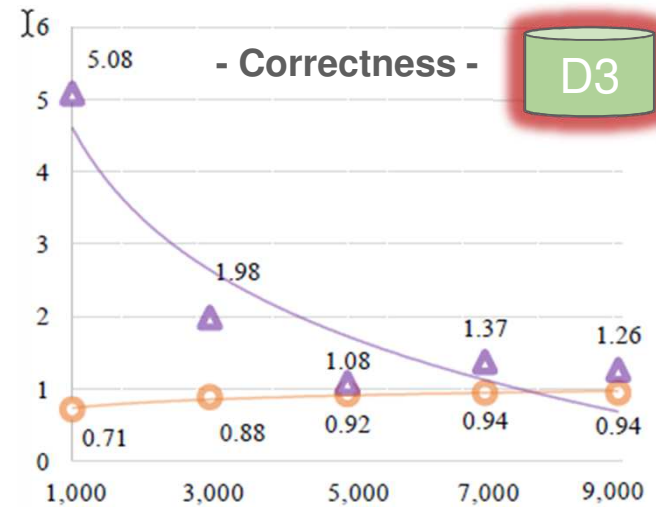
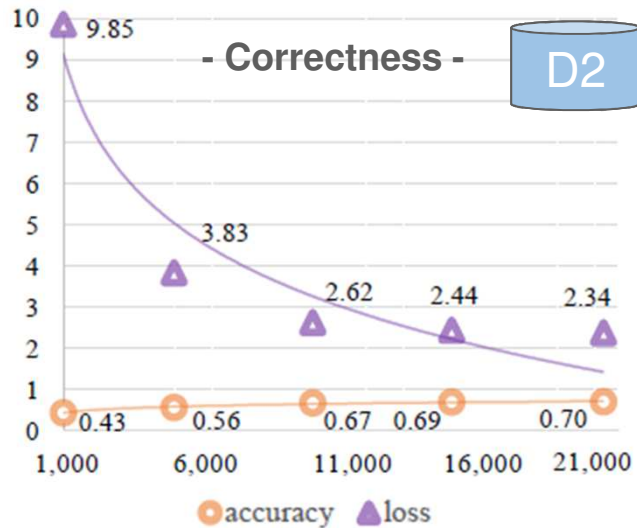
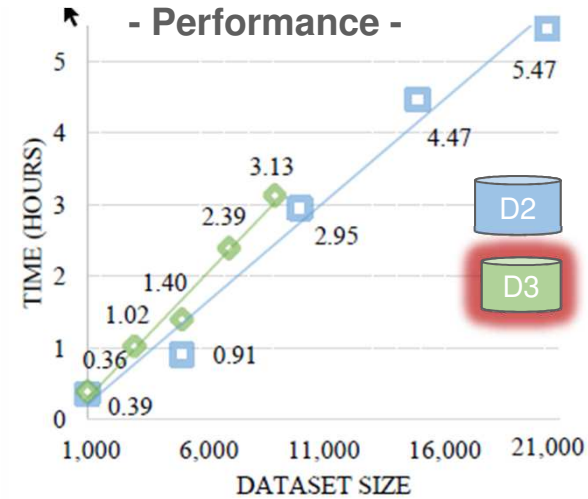
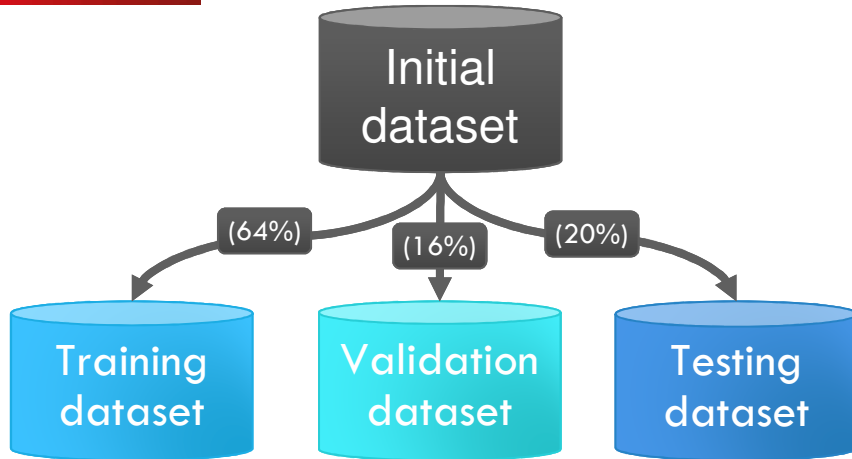
Adapted to output code.

About the training dataset of our PoC

- Where the examples come from?
 - Reverse-engineered Java code of the Eclipse IDE into a Java model.
→ MoDisco (<https://www.eclipse.org/MoDisco>).
 - Abstract the Java model to a pure high-level UML model by removing all the “low-level” details such as method implementations.
- Pre-processing step:
 - Model and code converted into an AST,
 - Variable renamed (Dictionary issue).
 - **Dataset (D1) contains 25,375 pairs of examples.**
- Cleaning step – discard pairs where the size of classes is too high:
 - Either raise RAM memory issue,
 - Or big size leads to the problem of long-term dependencies.
 - **Curated dataset (D2) contains 20,840 pairs of examples.**
- Cleaning step-bis – coding rule variations!
 - e.g., Inheritance may involve diversity with owners of getters and setters.
 - **New curated dataset (D3) contains 8,937 pairs of examples.**

Note: if for the same input, ANN receive different outputs (which is usually the case when writing code), they follow the “rule” which they have seen more often.

PoC, about correctness & performance



Conclusions and future work related this theme

- **Demonstrate the feasibility and interest...**



- Needs for high-quality examples... (as usual for ANN...),
- Dictionary problem,
- No support for operations on strings from ANN.



- Numerous examples reversed from open-source projects,
- Good execution performance of AI-based code generators,
- Acceptable training performance.

- **Next steps for us:**

- Experiment with transfer learning to enable reuse of trained networks in new projects where the styling guidelines may be slightly different,
- Extend the network capabilities to cover the generation of basic behavioral code.
- Continue to monitor the advance in AI and try out new features...

The two main pillars of MBE



B. Selic, "Model-driven development: its essence and opportunities," in Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06), 24-26 April 2006, Gyeongju, Korea.

What is the problem...



IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 39, NO. 6, JUNE 2013

What Industry Needs from Architectural Languages: A Survey

Ivano Malavolta, Patricia Lago, Senior Member, IEEE, Henry Muccini, Member, IEEE

languages, and tools in certain (research) topics, as needed by their users. The strengths, weaknesses, and specifically to software architectures, and specifically to languages have been introduced by the research in the user's perceived needs in architectural modeling industry. The user survey by support is asked to fill in a questionnaire of set 1 while practitioners are generally satisfied in the architectural language analysis features. In practice, many originate from industrial requirements of an architectural language.

empirical study

al issue is the proliferation of languages for software (SA) description without a clear of their merits and limitations. Terms of languages (ALs) can be found today, each with slightly different conceptual architectural notations, or semantics. They focus on a specific operational domain, some do not analysis while others support different is observed in [12], one of the reasons of so many architectural languages is frequent stakeholder concern. A language design decision involves judgment from the stakeholders. Stakeholder concerns evolving, and adapting to changing. Hence, it is difficult to capture all a single, narrowly focused notation, a unique language for notation, we must accept the end users for SA modeling, allow different types of stakeholder concern, modeling architecture (e.g., [11]) a standard and uniquely identified

Model-driven Development of Complex Software: A Research Roadmap

Robert France, Bernhard Rumpe



Robert France is a Professor in the Department of Computer Science at Colorado State University. His research focuses on the problems associated with the development of complex software systems. He is involved in research on rigorous software modeling, on providing rigorous support for using design patterns, and on separating concerns using aspect-oriented modeling techniques. He was involved in the Revision Task Forces for UML 1.3 and UML 1.4. He is currently a Co-Editor-in-Chief for the Springer International Journal on Software and System Modeling, a Software Area Editor for IEEE Computer and an Associate Editor for the Journal on IEEE Testing, Verification and Reuse.

Computer Languages, Systems & Structures 4:3 (2013) 139-155

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

Journal homepage: www.elsevier.com/locate/colse



Model-driven engineering conceptual model

Alberto Rodrigues da Silva
INESC-ID Institute Superior Technic - Universidade Nova de Lisboa

ARTICLE INFO

Article history:
Received 14 November 2014
Received in revised form 19 May 2015
Accepted 6 June 2015
Available online 23 June 2015

Keywords:
Modeling language
Micro-model
Software system
Model-driven engineering
Model-driven approach

Future
0-7696

1. Introduction

A model is an abstraction of a system representing a partial and simplified view of the system and its behavior. It is used to represent and understand the system as well as other areas such as Physics, Maths, etc. Models are used in the context of both thus language-based in nature and need Mathematics which are understood as in Models allow sharing a common vision promoting the communication among it while providing a more appropriate view according to objective criteria [8,9].

E-mail address: alberto@inesc-id.iscte.ucp.pt
http://dx.doi.org/10.1016/j.colse.2015.06.001
1473-044X/2015 The Author. Published by Elsevier
http://dx.doi.org/10.1016/j.colse.2015.06.001

Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem?

Jon Whittle¹, John Hutchinson¹, Mark Roundfield¹, Håkan Burdén², and Rogardt Hekla³

¹ School of Computing and Communications, Lancaster University, Lancaster, UK
² Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden

Abstract. An oft-cited reason for lack of adoption of model-driven engineering (MDE) is poor tool support. However, studies have shown that adoption problems are as much to do with social and organizational factors as with tooling issues. This paper discusses the impact of tools on MDE adoption and places tooling within a broader organizational context. The paper revisits previous data on MDE adoption (19 in-depth interviews with MDE practitioners) and re-analyses the data through the specific lens of MDE tools. In addition, the paper presents new data (20 new interviews in two specific companies) and analyzes it through the same lens. The key contribution of the paper is a taxonomy of tool-related considerations, based on industry data, which can be used to reflect on the tooling landscape as well as inform future research on MDE tools.

Keywords: model-driven engineering, modeling tools, organizational change.

1 Introduction

When describing barriers to adoption of model-driven engineering (MDE), many authors point to inadequate MDE tools. Den Haan [1] highlights “incomplete tools” as one of the eight reasons why MDE may fail. Kuhn et al. [2] identify five points of friction in MDE that introduce complexity; all relate to MDE tools. Sharon [3] found that “technology maturity [may] not provide enough support for cost-efficient adoption of MDE.” Tonussenti et al.'s survey reveals that 30% of respondents see MDE tools as a barrier to adoption [4].

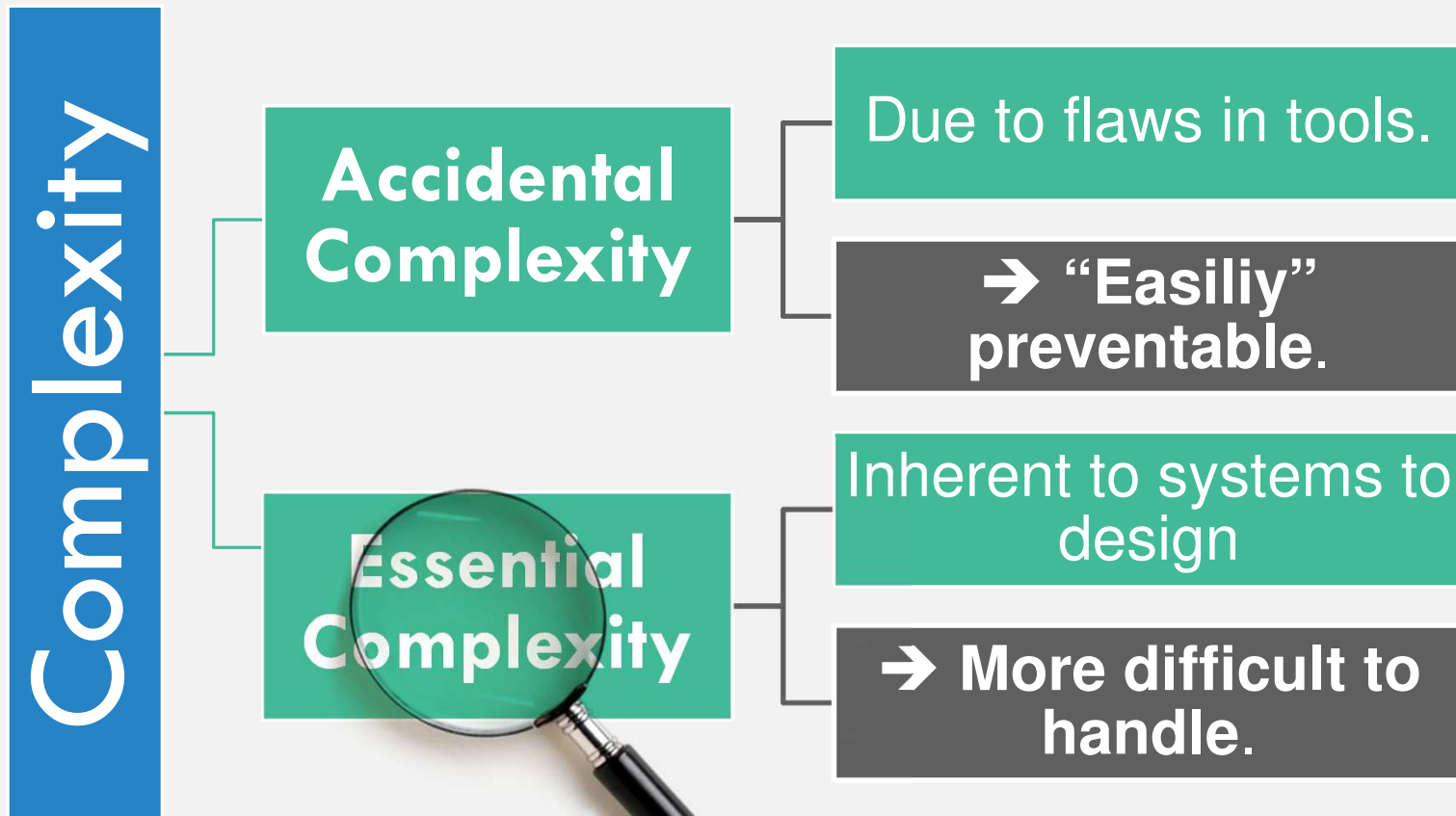
Clearly, then, MDE tools play a major part in the adoption (or not) of MDE. On the other hand, as shown by Hutchinson et al. [5], barriers are as likely to be social or organizational rather than purely technical or tool-related. The question remains, then, to what extent poor tools hold back adoption of MDE and, in particular, what aspects – both organizational and technical – should be considered in the next generation of MDE tools.

The key contribution of this paper is a taxonomy of factors which capture how MDE tools impact MDE adoption. The focus is on relating tools and their technical features to the broader social and organizational context in which they are

A. Moreira et al. (Eds.), MODELS 2013, LNCS 8107, pp. 1–17, 2013.
© Springer-Verlag Berlin Heidelberg 2013

Slidebar on complexity

Frederick. P. Brooks Jr., “No Silver Bullet Essence and Accidents of Software Engineering”, Computer, vol. 20, no. 4, pp. 10–19, Apr. 1987.



Our use case: system architecture



Conceptual modeling, a key step of system architecting.



Augment tools with AI-features to face essential complexity of conceptual modeling.



Empower experience if you are a debutant, or creativity if you are experienced.



→ Empower analogy thinking
by exemplification ←

What is an assistant?

Assistance: The Work Practices of Human Administrative Assistants and their Implications for IT and Organizations

Thomas Erickson, Catalina M. Danis, Wendy A. Kellogg, Mary E. Helander
IBM T. J. Watson Research Center
IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY
{snowfall | danis | wkellogg | helandm}@us.ibm.com

ABSTRACT

Assistance – work carried out by one entity in support of another – is a concept of long-standing interest, both as a type of human work common in organizations and as a model of how computational systems might interact with humans. Surprisingly, the perhaps most paradigmatic form of assistance – the work of administrative assistants or secretaries – has received almost no attention. This paper reports on a study of assistants, and their principals and managers, laying out a model of their work, the skills and competencies they need to function effectively, and reflects on implications for the design of systems and organizations.

Author Keywords

Administrative assistant, secretary, personal assistant, assistant, intelligent assistant, articulation work

ACM Classification Keywords

H.5.3 [Group and Organization Interfaces]: Computer-supported cooperative work, Organizational design, Theory and models; H.4.1 [Office Automation]: Time management (e.g., calendars, schedules), Workflow management; H.1.2 [User/MachineSystem]: Human factors, Human information processing; General Terms: Design, Human factors, Theory

INTRODUCTION

This paper is concerned with the work of assistance, work carried out in support of another's work. While it is quite common for one person to help another in passing – I might proofread a colleague's paper, or a friend might forward an article that she knows fits my interests – we are concerned with the case in which the majority of a person's work is in direct support of another person. Assistance in this sense is generally provided in the context of institutionally or culturally sanctioned roles, common examples being administrative assistants or, in the trades, apprentices.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CSCW'08, November 8-12, 2008, San Diego, California, USA.
Copyright 2008 ACM 978-1-60558-007-4/08/11...\$5.00.

Assistance is relevant to information technology in several ways. Most obviously, it serves as a model and metaphor for human computer interaction. The concept of an "intelligent assistant" (also known as "intelligent agents," "personal digital assistants" and "electronic secretaries"), has been extant in the information technology literature for decades. While sometimes this use of language is no more than an empty if provocative metaphor, other times it represents real if visionary ambitions. Perhaps the best known example is Apple Computer's Knowledge Navigator video [1], starring "Phil," an intelligent agent who scheduled meetings, reminded his principal of events, and handled phone calls with aplomb. In a more staid example Gutierrez and Hidalgo [10] wrote about their aim to create an intelligent assistant that "will remove much of the burden of administrative chores from its human user and provide guidance, advice, and assistance in problem solving and decision making." (p 126) More generally, any search of the information technology literature over the last decades will reveal a plethora of papers that describe "assistants" for programming, teaching, training, et cetera.

Assistance is also relevant to information technology in more literal ways. As its popularity as a model for human computer interaction attests, assistance is an important and pervasive type of human collaboration. The one-on-one form of assistance embodied by administrative assistants represents a common type, and is particularly interesting because its long term and in depth nature allows the development of collaborative practices and artifacts that are tailored to the particularities of a relationship and situation. More generally, viewed as a type of work, assistance plays a role in many forms of workflow which are not necessarily transactional, sequential or linear in nature, and in the structuring of work and communication processes at the organizational level. A better understanding of assistance at this level can offer insights to those charged with designing workflows, services and organizational structures.

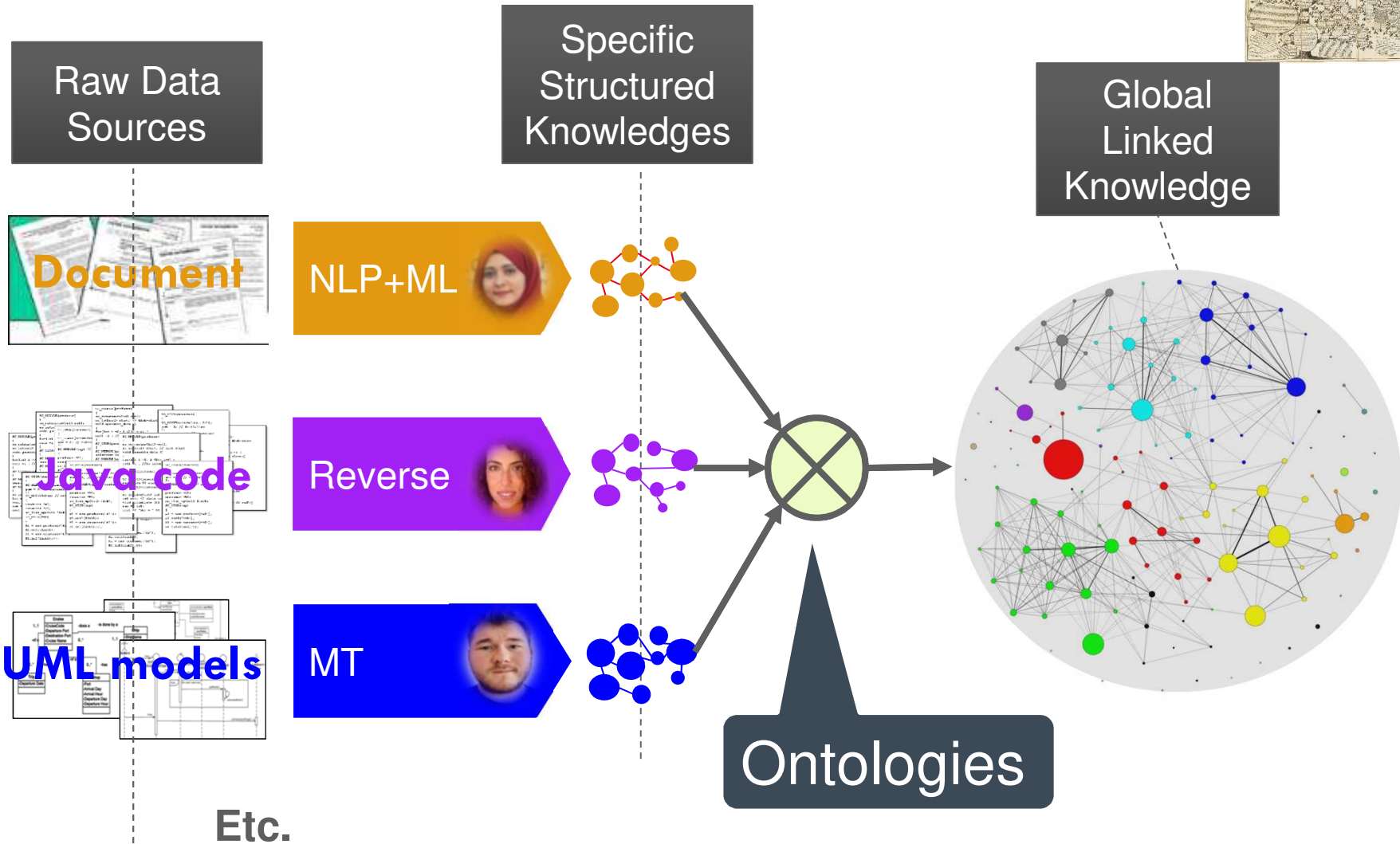
However, in spite of the long history of assistance as a model for human computer interaction, and the importance of assistance in the daily life of organizations, there is, as we shall see, little research that focuses on what administrative assistants actually do or how they go about doing it. The goal of this paper is to redress this situation.

“Assistants must **handle a stream of events** by reacting to it while **maintaining situational awareness** and **consolidating background knowledge**.”

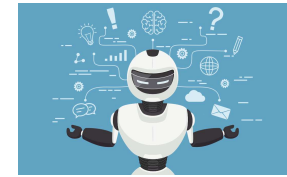
RQ1: How to get the background knowledge?

RQ2: when and how best to interact with the tool user?

RQ1: how to get the background knowledge



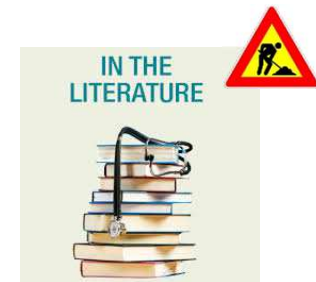
RQ2: when and how best to interact with the tool user?



- **RQ2.1 – When the bot should interact with modellers?**
 - Rely on the bot capacities to understand the activities of the modeller
 - **Bot awareness**
 - Our solution → the Awareness theory of Endsley [2]:
 - **Identity**: who is assisted (its profile, experience,...)?
 - **Authorship**: who is the origin of the model currently managed?
 - **Actions**: what the modeler is doing?
 - **Artifacts**: what are the related resources?
 - **Intention**: what is the goal of the modeler?



- **RQ2.2 – How the bot should present the information to modellers?**
 - Ongoing SLR focused on:
 - Visualization information,
 - Distributed cognition (i.e., where the info should be presented),
 - And cognitive dimensions of the assistance.



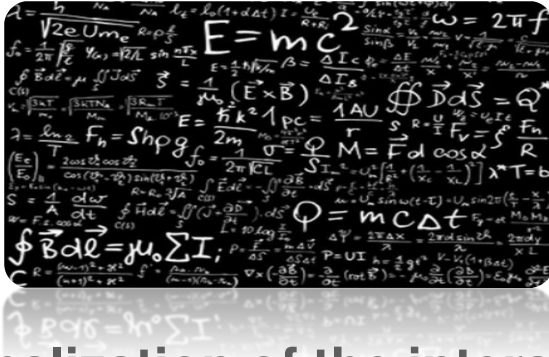
[1] P. Dourish and V. Bellotti, "Awareness and coordination in shared workspaces", CSCW '92, Toronto, Canada, 1992.

[2] M. R. Endsley, "Design and evaluation for situation awareness enhancement", Human Factors Society 32nd Annual Meeting, Santa Monica, CA, 1988.

Cinema break on an Architecture Modeling Bot...

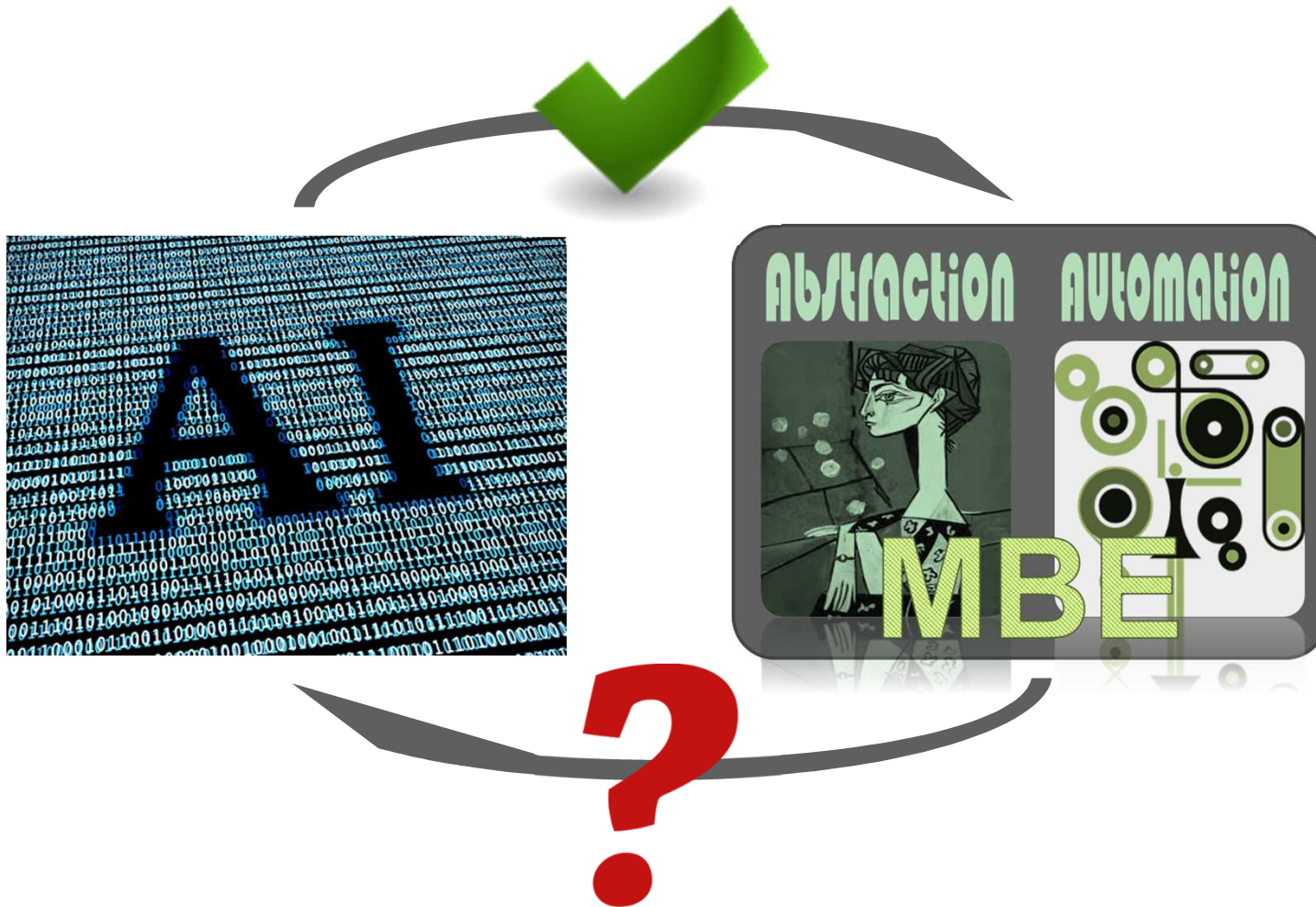


Remaining challenges we want to focus on



- Formalization of the interactions between modelling User & Bot in the context of MBE.
- A model-based methodology to design and embed modeling bots into modeling IDE.
- How to create knowledge from data...?
- An framework that support Awareness of modeling workspace.
- A modeling bot focusses on architecture concerns in Papyrus.
- An information integration bot that can create & consolidate knowledge on its own from multiple & heterogenous data sources.

AI for MBE, and reversely?



AI are “black-box”!

- Modeling and analyzing AI-based systems typically requires accepting some uncertainties about their precise behavior.
- However, trust in AI is a key point for their acceptance and safe deployment.

→ Needs to “model and operate” this uncertainty!



Premier international conferences on
Uncertainty in Artificial Intelligence

But what is "Uncertainty modeling" ?



- <https://www.bipm.org> -

GUM definition*: "the quality or state that involves imperfect and/or unknown information".

* JCGM 100:2008, Evaluation of measurement data - Guide to the expression of uncertainty in measurement (GUM), Joint Com. for Guides in Metrology, 2008.

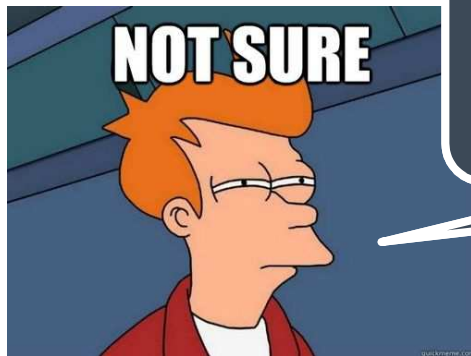
- Three kinds of uncertainty:

- Measurement uncertainty
 - Inability to know with complete precision the value of a quantity (e.g., $x = 3.0 \pm 0.01$).
- Occurrence uncertainty
 - Likelihood that a physical entity represented in a model actually occurs in reality.
- Belief uncertainty
 - Situation where one is uncertain about a statement made about a knowledge or a system, or even the environment of a system.



L. Burgueño, R. Clarisó, J. Cabot, S. Gérard, and A. Vallecillo, "Belief Uncertainty in Software Models," in 2019 IEEE/ACM 11th International Workshop on Modelling in Software Engineering (MiSE), 2019.

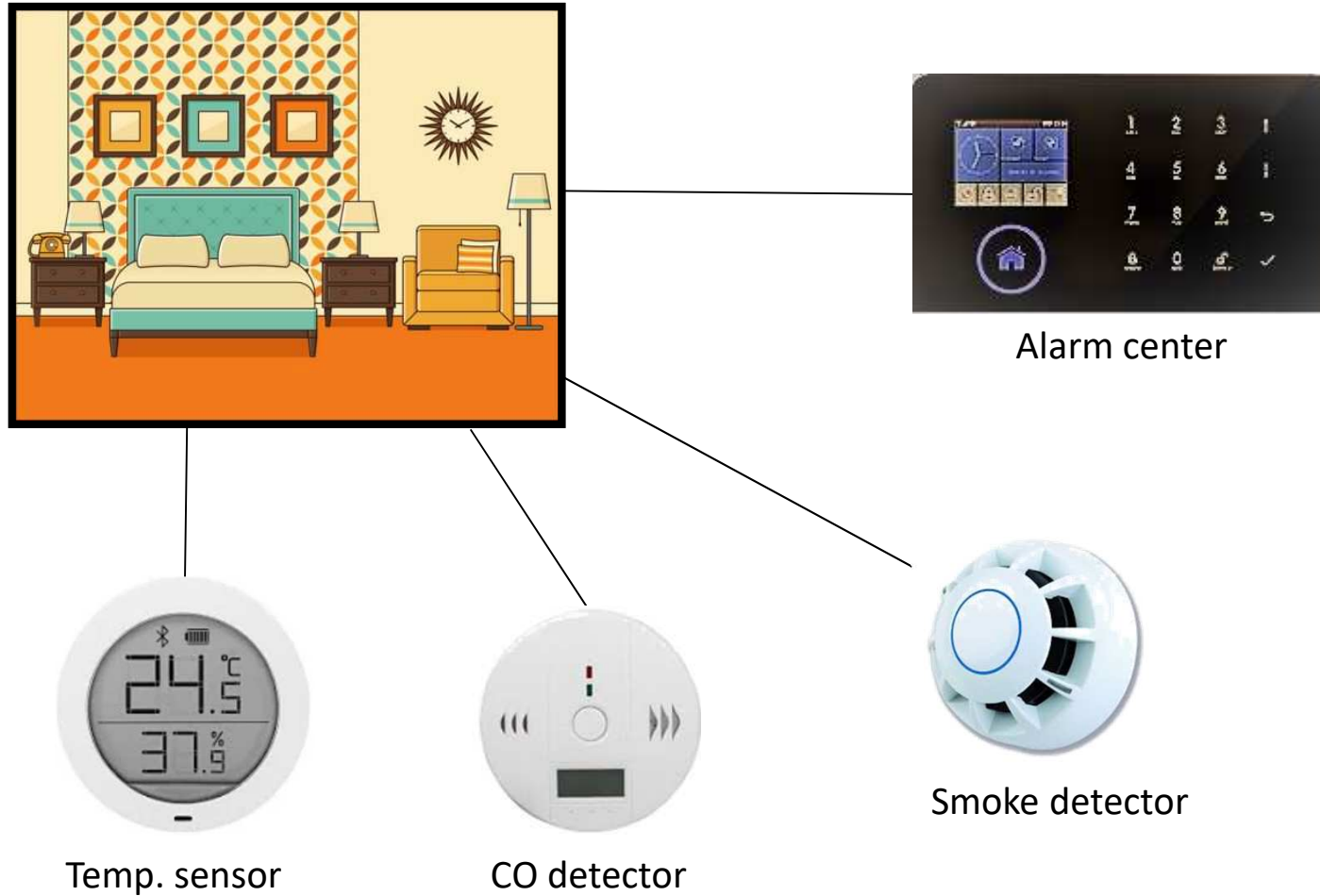
(Belief statements are usually subjective!)



Not sure my learning source is fully accurate!
Not sure how far my algo is precised!
etc.

- **This work is hence aiming at answering the following questions:**
 - **RQ1** - How to specify the belief uncertainty of data?
 - **RQ2** - How to integrate this feature in existing modeling language?
 - **RQ3** - How to use the information about the uncertainty?

Our motivating example...



Some belief statements on our example

- The CO and smoke detectors that we bought have a reliability of 90% (i.e., 10% of their readings are not meaningful). → *Precision of the values*
- We can only be 98% sure that the precision of the temperature sensor is 0.5°C, as indicated in its datasheet. → *Uncertainty of the values*
- We are 95% confident that the presence of high temperature, high CO level and smoke really means that there is a fire in the room. → *About the behavioral rules*

➡ Paul only assigns a credibility of 50% to the operations that indicate if the room is hot or cold. In contrast, Marie thinks they are 99% accurate. → *Individual belief agents*

- Room #3 is close to the kitchen and frequently emits alarms. Everybody thinks that 90% of them are false positives. → *Individual instances*
- Paul doubts that the type of attribute “number” of class “Room” is Integer. He thinks it may contain characters different from digits. → *About the model itself: types*

➡ Marie is unsure if an “AlarmCenter” has to be attached to only one single Room. She thinks they can also be attached to several. → *About the model itself: relations*



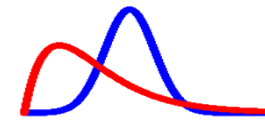
How to represent these uncertainties?
How to integrate them into the system models?

RQ1 - How to specify the belief uncertainty?

Different stakeholders may have different estimation.

- Goal => be able to assign degree(s) of belief to model statements.
- Solution => Bayesian probabilities while being a classical model for quantifying subjective beliefs .

Resp1 - Credence to measure belief uncertainty



Note: credence is a statistical term that refers to a measure of belief strength: used here to express how much an agent believes that a proposition is true (e.g, a modeler can be 99% sure the type used to represent a given property is correct.).

RQ2 - How to integrate this feature in existing modeling language?

Resp2.a - Extend OCL/SysML/UML datatypes to integrate information about the uncertainty of their values.

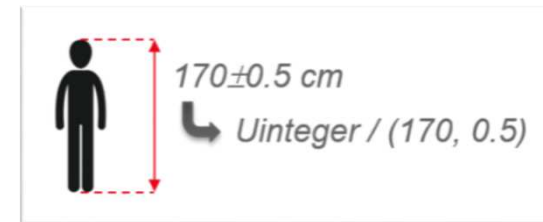
- Standard datatype extensions e.g., UInteger, and UReal.

- Represented by a pair

(x, u)

expected value

uncertainty



- UBoolean values uncertainty does not refer to measurement uncertainty, but to confidence.

- Represented by a pair

(b, c)

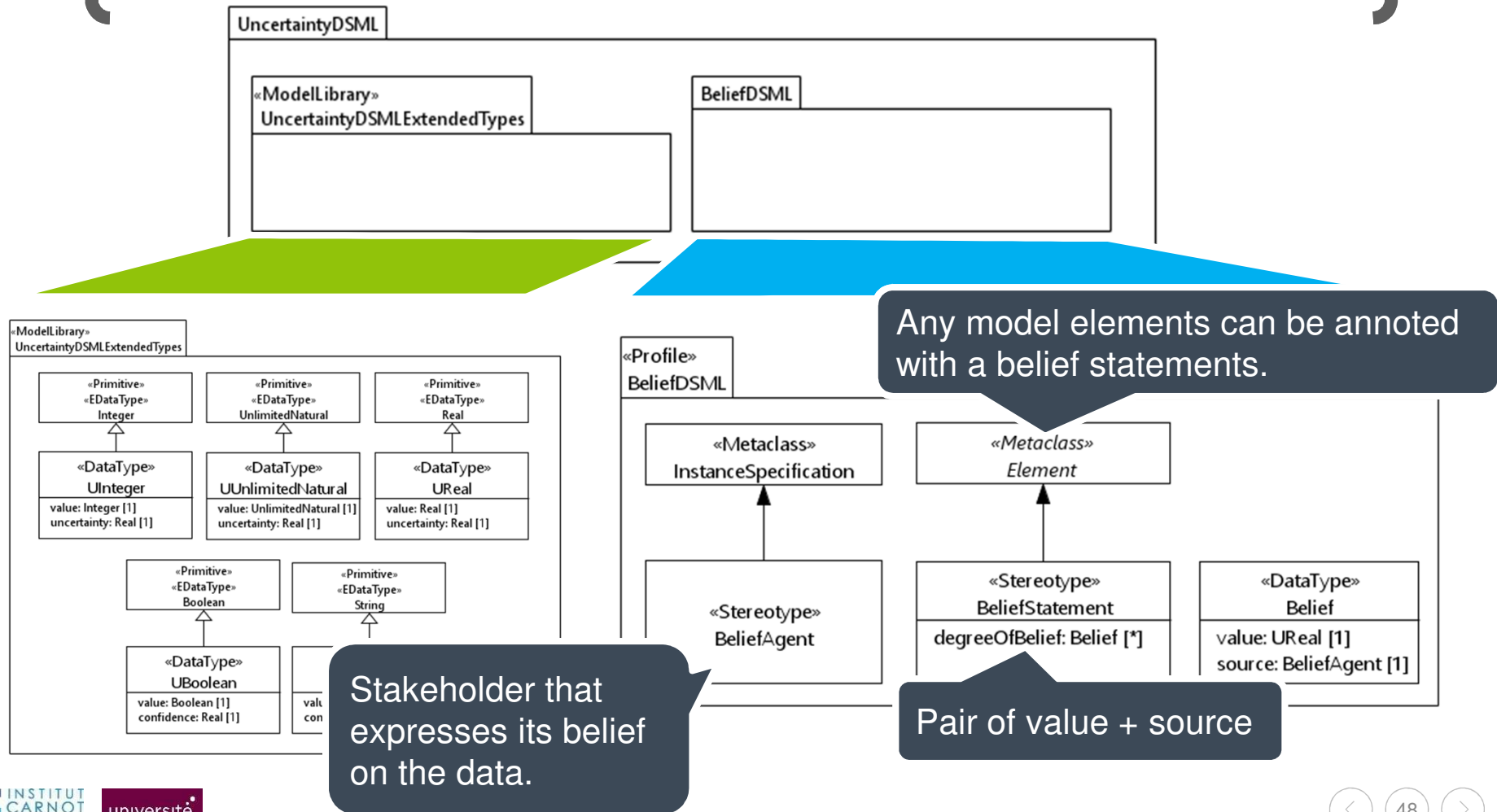
boolean value →
true or false

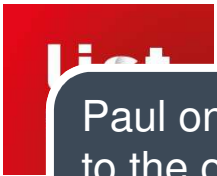
confidence that b is
certain → Real number
in the range [0..1]



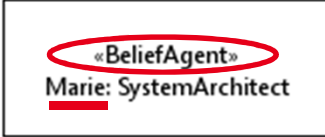
RQ2 - How to integrate this feature in existing modeling language? (Con't)

Resp2.b – Define a UML profile for uncertainty modeling (can be applied on both UML and SysML models).



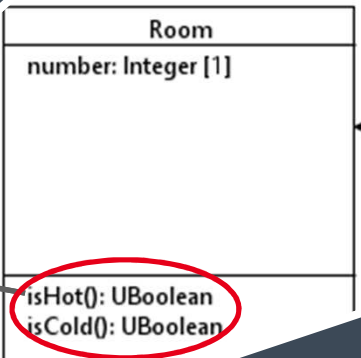


Belief Agent Definition



Definition of the Belief Agents for our system.

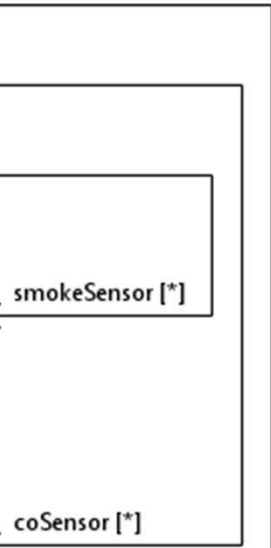
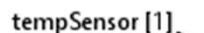
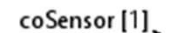
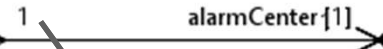
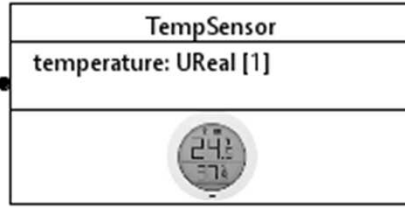
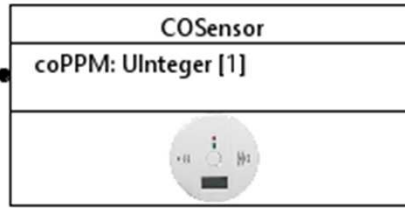
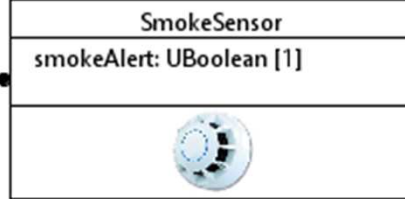
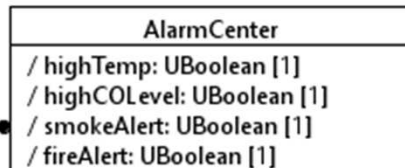
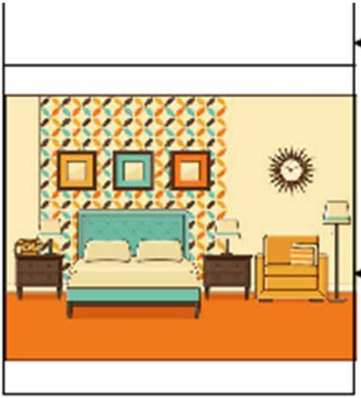
Paul only assigns a credibility of 50% to the operations that indicate if the room is hot or cold. In contrast, Marie thinks they are 99% accurate.



«BeliefStatement»
{(Marie, 0.99),
(Paul, 0.5)}

«BeliefStatement»
{(Marie, 0.5)}

Marie is unsure if an "AlarmCenter" has to be attached to only one single Room.



Uncertainty Modeling: executive summary and future work

- **Explicit representation and management of belief uncertainty in software models...**

in terms of degrees of belief assigned to model elements by separate belief agents...

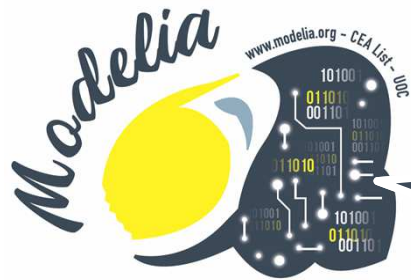
and about the credibility of:

- The values of the represented elements,
 - The measurement uncertainty of these values,
 - And the way in which we have modeled the system (e.g., types of the attributes, types of relationships and their cardinalities).
- **Future work**
 - Contribute to standard:
 - The OMG is working towards a metamodel for the Precise Specification of Uncertainty Modeling (PSUM)
 - Associating evidences to belief statements,
 - Representing degrees of beliefs in other types of models (use cases, sequence diagrams, pre- and postconditions, ...),
 - And investigate further application domains (e.g., model inference, model assistant).



And what about next?

Conclusions and next steps...



“Continue monitoring AI progresses and our experiences with AI.”

- Modelia projects:



Loli Burgueño, post-doc (2019-2020) on “**Uncertainty & AI-4-MT**”.



Maxime Savary-Leblanc, Phd student (2019-2022) on “**Modeling bots**”.



Takwa Kochbati, Phd student (2019-2022) on “**From Text to Conceptual Models**”.



Edouard Batot, post-doc (2020-2022) on “**Traceable co-evolution management by ML**”.





An finally...



Loli Burgueño, post-doc
(2019-2020) on
“**Uncertainty & AI-4-MT**”.

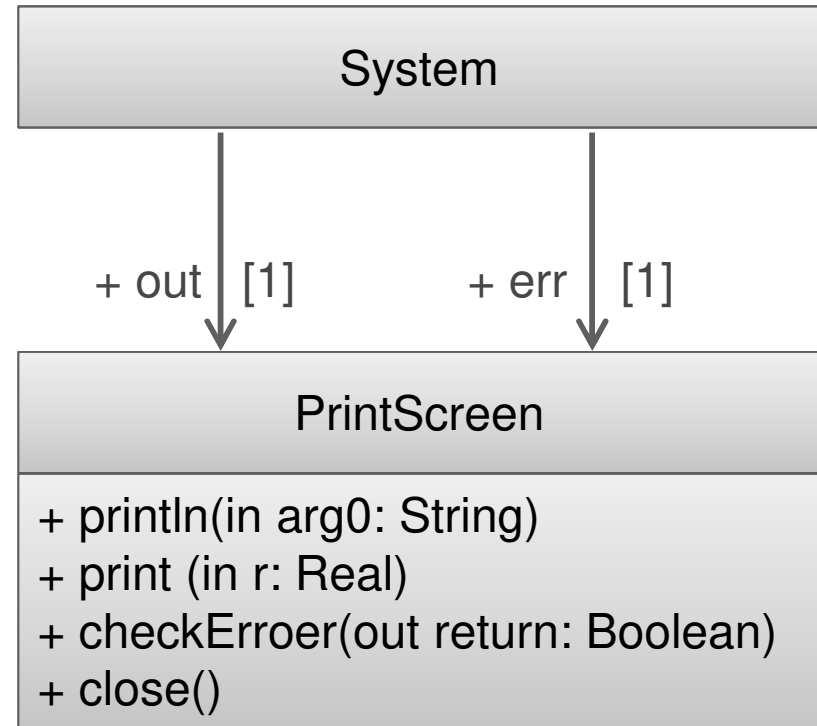
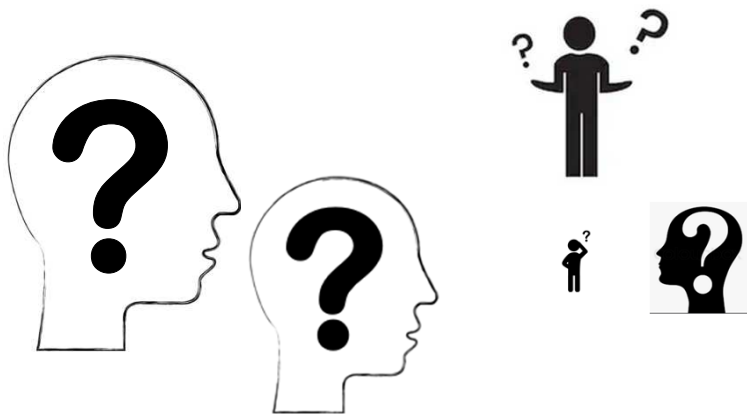
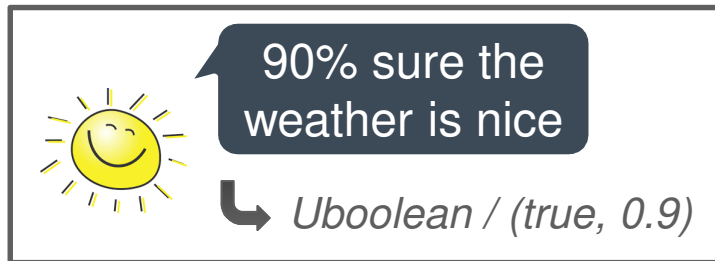


Maxime Savary-Leblanc,
Phd student (2019-2022) on
“**Modeling bots**”.



Commissariat à l'énergie atomique et aux énergies alternatives
Institut List | CEA SACLAY NANO-INNOV | BAT. 861 – PC142
91191 Gif-sur-Yvette Cedex - FRANCE
www-list.cea.fr

Établissement public à caractère industriel et commercial | RCS Paris B 775 685 019



usability
interoperability
workflow integration
 performance accessibility
 high skill **complexity**
steep learning curve

