Model-Aware Software Engineering
A Knowledge-based Approach to Model-Driven Software Engineering

Robert Andrei Buchmann[1], Mihai Cinpoeru[1], Alisa Harkai[1] and Dimitris Karagiannis[2]
[1]*Business Informatics Research Center, Babeş-Bolyai University, str. Th. Mihali 58-60, Cluj Napoca, Romania*
[2]*Research Group Knowledge Engineering, University of Vienna, Waehringerstr. 29, Vienna, Austria*

Keywords: Agile Modelling Method Engineering, Resource Description Framework, Model-Driven Software Engineering, Knowledge Representation.

Abstract: Standard modelling languages enabled the Model-Driven Software Engineering paradigm, allowing the development of model compilers for code generation. This, however, induces a *subordination of implementation to the modelling language*: the modelling benefits are confined to a fixed semantic space. On the other hand, the rise of agile software development practices has impacted model-driven engineering practices - an Agile Modelling paradigm was consequently introduced. This was later expanded towards the Agile Modelling Method Engineering (AMME) framework which generalizes agility at the modelling method level. By observing several AMME-driven implementation experiences, this paper specialises the notion of Model-Driven Software Engineering to that of *Model-Aware Software Engineering* – an approach that relies on modelling language evolution, in response to the evolution of the implemented system's requirements. The key benefit is that the modelling language-implementation dependency is reversed, as the implementation needs propagate requirements towards an agile modelling language.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

The convergence of agile software development practices and the Model-Driven Engineering paradigm has naturally lead to the Agile Modelling methodology (Ambler, 2002). However, agility in AM focuses on modelling practices rather than modelling methods.

The underlying assumption is that all software development needs can be subordinated and conceptually subsumed to the fixed semantic space defined by standard modelling languages. This is a reasonable assumption in the two dominant perspectives on the role of conceptual modelling in software engineering: (i) the *modelling is documenting* perspective, where models act as guidance for developers, and are therefore "distilled" by a human programmer relying on consensus with respect to notation, structure and meaning; (ii) the *modelling is programming* perspective, where models are input for code generators that have been preprogrammed based on the fixed semantic space.

The *Agile Modelling Method Engineering* (AMME) framework (Karagiannis, 2015) introduced a third perspective, that of *modelling is knowledge representation*, where the modelling language is tailored for capturing with diagrammatic means the enterprise knowledge that is relevant for implemented artefacts. The relation to implemented artefacts is not based on code generation, but rather on parameterization of software artefacts with properties that are extracted or inferred from models. Compared to *process-aware information systems* (van der Aalst, 2009), AMME advocates a full customization of the modelling language.

This vision of AMME was adopted for the purposes of the work at hand, which focusses on the implementation of project-based instances of a Model-Aware application for the goals of (i) evaluating the feasibility of this software engineering methodology and of (ii) positioning AMME in a novel, knowledge-driven software development method. Therefore the proposed contribution is ther a software development method (labelled as *Model-Aware Software Engineering*) that assumes the adoption of AMME for modelling activities, as an alternative to the standards that traditionally drive model-driven engineering efforts.

The remainder of the paper is structured as follows: Section 2 provides background on AMME.

Buchmann, R., Cinpoeru, M., Harkai, A. and Karagiannis, D.
Model-Aware Software Engineering.
DOI: 10.5220/0006694102330240
In *Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2018)*, pages 233-240
ISBN: 978-989-758-300-1
Copyright © 2019 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

Section 3 shows how AMME fosters knowledge processes that establish the "knowledge-driven" quality of the hereby proposed method. Section 4 employs an illustrative example that is currently being evolved in project-based work. Section 5 comments on related works. The paper ends with a concluding evaluation and outlook.

2 BACKGROUND ON AGILE MODELLING METHOD ENGINEERING

The keynote paper of (Karagiannis, 2015) introduced the principles of Agile Modelling Method Engineering, indicating the Open Models Laboratory - OMiLAB (Open Models Laboratory, 2017) as a deployment environment and architecture where AMME emerged and is being evaluated in project work involving the development of domain-specific modelling tools - see a presentation of tools in (Karagiannis et al., 2016). A commonly employed platform for the fast prototyping of modelling tools is ADOxx (BOC, 2017). Earlier AMME experience reports are available in the literature (Buchmann and Karagiannis, 2015) - however, such experiences focus on the development of modelling tools and not on the software development processes that can be supported by such tools and their agile qualities – a gap that we aim to fill through this paper.

The usable result of AMME is a **modelling tool** that deploys a modelling method, as defined in (Karagiannis and Kühn, 2002). Such a modelling tool evolves through iterations reflecting the increments applied on the modelling method's building blocks (i.e., the language, procedure and functionality). The fundamental drivers of this process are the **modelling requirements** – a specialized notion of requirements focusing on modelling scenarios / use cases. Just as requirements are considered unstable or evolving in agile software development, modelling requirements are also considered essentially unstable in AMME – several pragmatic reasons are behind this consideration, confirmed by this paper's work context: (i) users lacking in modelling experience will start raising change requests once they gain initial hands-on experience; (ii) if software is implemented based on the created models, the changing requirements for the software artefacts will propagate in modelling requirements.

3 PROBLEM STATEMENT AND SOLUTION OVERVIEW

The work at hand proposes a model-driven software engineering method that *reverses the traditional subordination between implementation and models* – i.e., instead of having the implementation subordinated to an invariant modelling language (i.e., confined to its fixed semantic space), the proposal is to have the modelling language subordinated to evolving implementation needs. Thus, a modelling language (and tool) should be agilely tailored and evolved through AMME to expose the semantics needed for implementation.

The proposed method expands AMME towards the goal of software engineering. The applied extensions are summarized in Figure 1 (i.e., the Data Management and the Implementation lane). Since AMME produces an evolving modelling language (and, consequently, an evolving semantic space) the core assumption of traditional Agile Modelling (and associated software development processes) that models comply with some consensus on structure and semantics does not hold anymore. It is not feasible to evolve code generators in synchronicity with the modelling language evolution – change requests may be as drastic as adding an entirely new type of diagram to the language, thus reusability is very limited. AMME sacrifices reusability for benefits pertaining to *specificity* –the modelling environment will act as a Knowledge Management System rather than as component in some standards-based roundtrip engineering cycle. Under these assumptions, AMME does not serve as a system design enabler, but rather as a *flexible knowledge acquisition enabler*. Since AMME is essentially a metamodelling framework, it supports the acquisition of knowledge on two levels: (i) *Domain knowledge*, captured in the metamodel tailored for each language iteration; (ii) *Case knowledge*, captured through the act of modelling (by using a language iteration designed on the previous level).

The two knowledge layers should be exposed to software development processes in an agile manner – i.e., changes in both the language and the model contents should be immediately made available to software development processes in a uniform representation that covers both layers. Models should be amenable to reasoning (i.e., if what was explicitly captured in diagrammatic form is deemed insufficient) - for this purpose, the Resource Description Framework (W3C, 2017a) is employed to streamline the knowledge conversion flow between the method's phases (Figure 1):

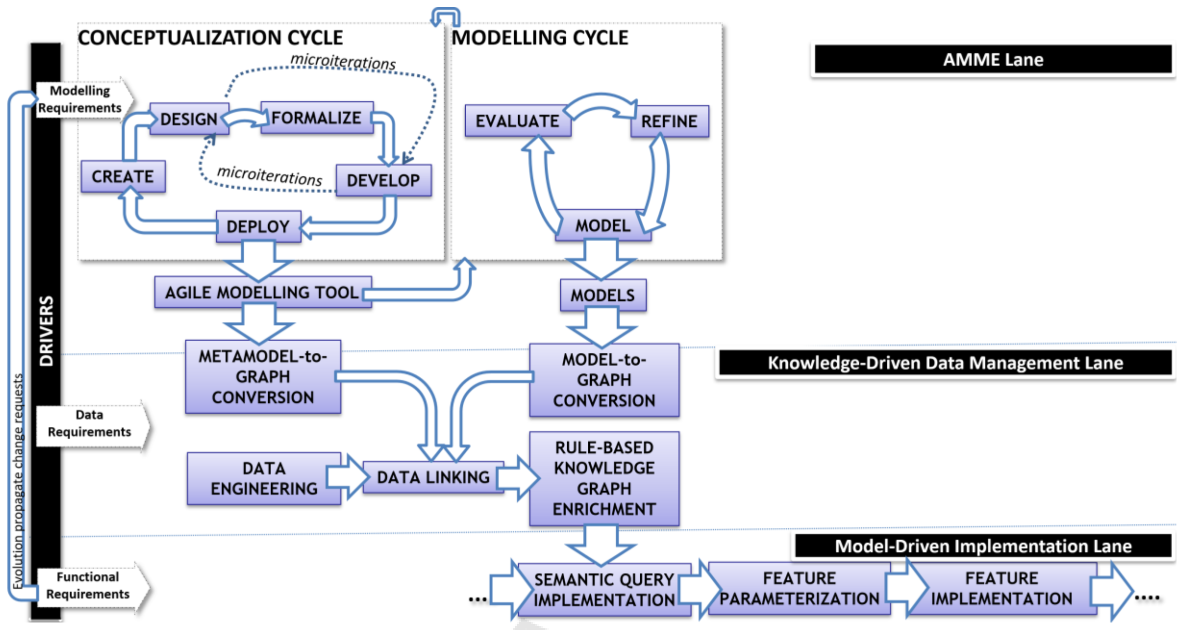


Figure 1: Knowledge conversion flow in Model-aware Software Engineering.

The Conceptualization Cycle takes as input requirements and domain knowledge to produce an *agile modelling tool* engineered through AMME's incremental and iterative cycle comprising the following phases: *Create* (knowledge acquisition and modelling requirements analysis), *Design* (specification of the modelling method building blocks, including metamodel), *Formalize* (method formalization for a targeted audience or goal), *Develop* (modelling tool implementation) and *Deploy* (modelling tool deployment and usage). Microiterations involving only the design and tool implementation are common for fast prototyping purposes; they are performed on metamodelling platforms, benefitting from built-in user interaction and model management features, thus allowing the engineer to focus on the modelling semantics. **The Modelling Cycle** uses the tool produced by the Conceptualization Cycle to define models along a typical modelling cycle. Feedback from hands-on experience will drive new iterations of the Conceptualization Cycle. In the **Knowledge-Driven Data Management Lane**, model contents, enriched with metamodel information are converted to RDF knowledge graphs and linked through Linked Data techniques to any data entities that are not depicted in models. A graph database with reasoning capabilities is necessary - GraphDB (Ontotext, 2017) has been used in implementations. An adapter was implemented for ADOxx, to serialize models (enriched by metamodel information), according to a highly abstract meta-metamodel easily translatable to other metamodelling platforms, according to some transformation rules detailed in (Karagiannis and Buchmann, 2016). Their output is a conglomerate of RDF graphs – one graph per diagram, including an RDF schema derived from the metamodel and metadata associated with each diagram. This "model base" can be further enriched by the reasoning engine of the database (GraphDB supports both custom rules and OWL axioms), thus further filling the semantic gap between model contents and front-end. The **Implementation Lane** covers the model-aware implementation tasks. The developed front-end artefacts are semantically parameterized with information retrieved via SPARQL queries (W3C, 2017b) from models, data, model-data links and any inferences that might have been executed on the "model base". Requirements for changes in the developed functionality can propagate back to the modelling language, thus triggering new AMME iterations, including the reprototyping of the modelling tool and the model base regeneration.

4 ILLUSTRATIVE EXAMPLE

4.1 Initial Requirements

Stakeholders require a Workflow Management System driven by diagrammatic process descriptions. Their organization coordinates a virtual enterprise offering customized clothing, where a network of candidate tailors, embroidery providers and delivery couriers can contribute, based on capacity and availability, to the execution of make-to-order production and delivery processes.

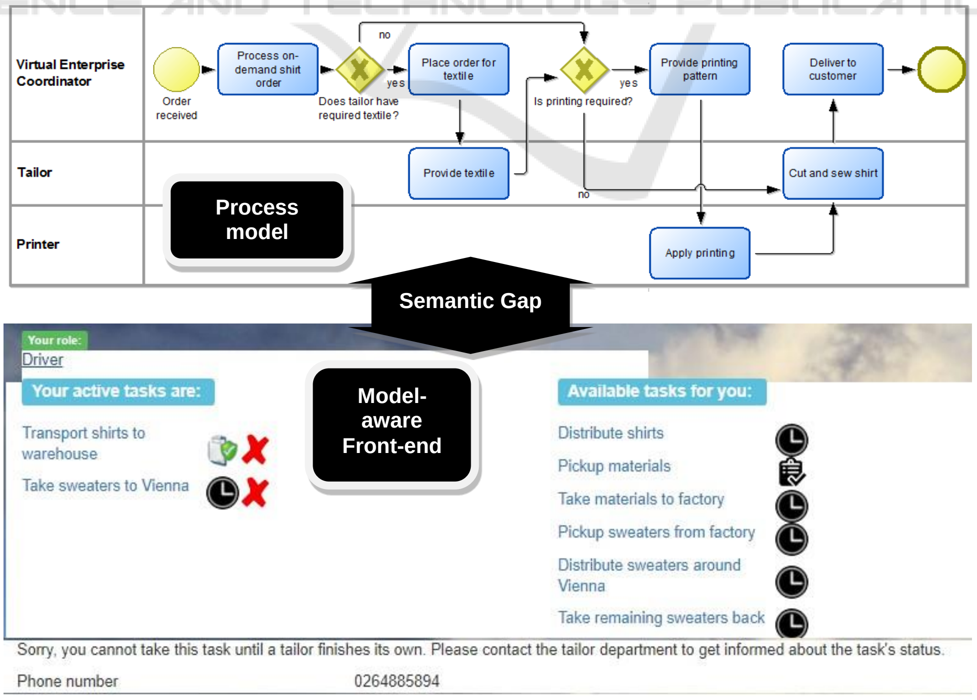
Figure 2 shows a simplified make-to-order production process together with a screenshot of task assignments in the workflow management system, showing the tasks of the authenticated user – active, fulfilled and pending (in the latter case showing contact data for those responsible). Between the required front-end functionality and the richness of the model information there is an obvious semantic gap that is commonly filled in Workflow Management Systems by the data model employed at run-time, using the model description as a backbone to guide the task flow. Conceptual redundancy manifests between the data model employed at run-time and the metamodel employed at design-time (e.g., the roles expressed by pools). Shifting conceptual fragments from the run-time data model towards the modelling environment will empower the modeller to drive process execution.

4.2 Advanced Iteration

The following modelling requirements are derived for an evolved iteration: **Semantic Requirements**: The virtual enterprise ecosystem must be described in more detail than what the swimlanes/pools allow. That is, dedicated concepts and relations must be devised to capture organizational structures, roles and instance employees as well as business partners grouped by the capability they can provide to the virtual enterprise. Similarly, a geographical coverage model should provide at least a grouping of targeted locations. Instance data properties may be necessary for those elements representing instances (e.g., address or coordinates for locations, contact data for business partners and employees). A distinction must be ensured between domain-specific task types (production tasks, driving tasks and delivery tasks). **Syntactic Requirements**: The new concepts must be separated from the process description in distinct types of models that may potentially evolve later independently of one another: one for business participants and one for locations, with the ability of mapping them to process tasks through hyperlinks across models. **Notational Requirements**: Simple groupings as graphical containers are preferable to visual connectors. The distinction between task types must be reflected visually. The notation should be enriched with domain specific visual cues rather than a standard notation. Visual cues should act as anchors for the hyperlinks between models.

A traditional model-driven system would assume that the modelling language is invariant. Consequently, such systems will assimilate any emerging semantic requirements in their run-time components and data model (i.e., the Task Manager functionality in a Workflow Management System).

SCIENCE AND TECHNOLOGY PUBLICATIONS



Sorry, you cannot take this task until a tailor finishes its own. Please contact the tailor department to get informed about the task's status.

Phone number	0264885894

Figure 2: Example of process and process-aware front-end.

Certain descriptions and properties implied by the aforementioned requirements are out of the scope of common business process modelling languages – e.g., an organizational chart, mappings between roles and instance candidates (as commonly necessary in virtual enterprise agile configurations), the geographical coverage of process executions or task assignments (typically performed in the Task Manager functionality).

It can be argued that some of this information belongs to the realm of instances and execution-time data repositories – however, the following aspects must be considered: (i) some data is sufficiently stable to be stored in models; (ii) certain instance representations are of interest for modelling in general (see also the multi-level modelling paradigm (Clark et al., 2014)) or for specific model analysis goals (e.g., workload simulation). Numerous types of models include elements that designate instances – e.g., concrete business partners, concrete locations, concrete software, As-Is or To-Be system components. Their properties can be just as stable as process models and therefore semantically coupled with more abstract model elements (a common situation in Enterprise Architecture Management).

In other words, those parts of a relational database that have a rather invariant nature and can support a modelling use case (e.g., simulation, model-based reporting) may be transferred to the model base, rather than being redundantly covered by both models and run-time components. Links between their model representations and their more dynamic properties (e.g., real-time availability of employees) will be maintained with the help of the inherent linking mechanisms provided by RDF.

Figure 3 shows a sample of how the BPMN process in Figure 2 can be evolved in a domain-specific modelling language with three types of diagrams – processes, locations and participants. Hyperlinks establish semantic relations between these models (e.g., locations of participants, task assignments on role level or instance level).

Such an evolution requires, of course, a reimplementation of the modelling tool, which is supported by the AMME conceptualization cycle and its fast prototyping support. Agility can manifest in all building blocks of the modelling method – *notation* (e.g., replacing the BPMN symbols with domain-specific visual cues), *syntax* (e.g., splitting the metamodel in multiple model types connected through hyperlinks), *semantics* (e.g., adding new concepts, specializing concepts, adding domain-specific property sheets), *functionality* (e.g., scripting model-driven functionality relevant for the current language iteration).

Further down the development process, the customized models are exported in an RDF knowledge base and subjected to relevant extensions. Figure 4 isolates a fragment that contains model elements from all three model types depicted in Figure 3, including the hyperlinks between them. It also depicts the enriched machine-readable graph that can be derived from it, by applying several graph extensions (e.g., OWL inferences or rules): (i) *Links to dynamic data* (e.g., real time *availability*) not included in models; such links rely on the URI identification scheme, with model elements having the same identifier as their counterparts in the external data model (assumed to be a semantic graph database, to simplify interoperability); (ii) *Inferred direct relations* (e.g., *directFollowedBy*) corresponding to the graphical connectors available in models (connectors are typically n-ary relations to also capture their annotations); (iii) Inferred types based on property restrictions (e.g., *AvailableTextileProvider* for those whose *availability* property is set to true and are contained within the *CandidateTextileProvider* box of *required capability*); (iv) Inferred types based on property restrictions (e.g., *AvailableTextileProvider* for those whose *availability* property is set to true and are contained within the *CandidateTextileProvider* box of *required capability*); (v) Certain relations (e.g., *assigned instance*) may be set either in the modelling user interface (e.g., the coordinator directly assigning instances to tasks through modelling means) or in the front-end interface (e.g, a user taking responsibility for tasks assigned to its capability container pool); (vi) Inferred relations based on relevant property chains (e.g., the *destination* identified by combining visual containment with instance availability).

Examples of such axioms are provided here for case (iii), with a richer discussion on OWL reasoning on model contents being available in (Karagiannis and Buchmann, 2018).

```
:containedBy owl:inverseOf :contains.

:TextileProvider
   owl:onProperty :containedBy;
   owl:hasValue :CandidateTextileProvider.

:AvailableInstance
   owl:onProperty :availability;
   owl:hasValue true .

:AvailableTextileProvider
   owl:intersectionOf
   (:AvailableInstance :TextileProvider).
```

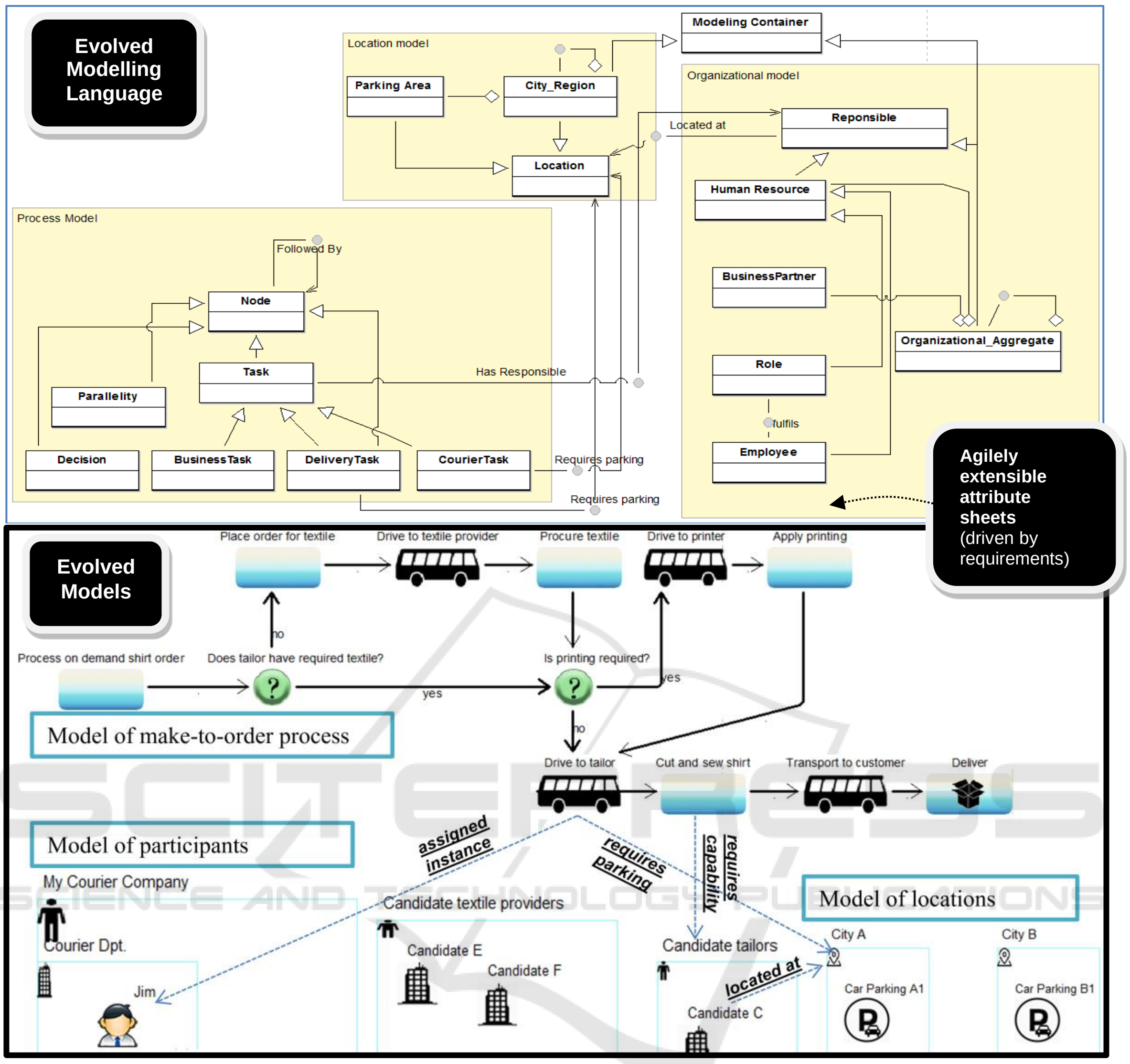


Figure 3: The agile modelling tool – metamodel (top) and model samples (bottom).

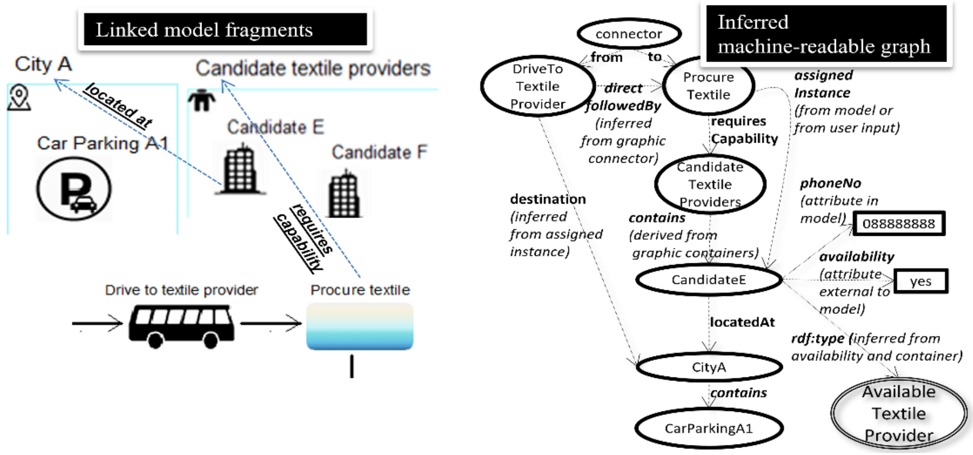

Linked model fragments
City A
Car Parking A1
located at
Candidate textile providers
Candidate E
Candidate F
requires capability
Drive to textile provider
Procure textile
Inferred machine-readable graph
connector
from
to
DriveTo Textile Provider
direct followedBy (inferred from graphic connector)
Procure Textile
assigned Instance (from model or from user input)
requires Capability
Candidate Textile Providers
destination (inferred from assigned instance)
contains (derived from graphic containers)
phoneNo (attribute in model)
088888888
availability (attribute external to model)
yes
CandidateE
locatedAt
rdf:type (inferred from availability and container)
CityA
contains
CarParkingA1
Available Textile Provider

Figure 4: Machine-readable knowledge graph derived from inferences applied on model content.

5 RELATED WORKS

The proposed software engineering method converges from a tradition in investigating flexibility in semantics-driven engineering methods – see the paradigms of situational method engineering (Kumar and Welke, 1992), ontology engineering (Corcho et al., 2003) and agile software engineering (Agile Manifesto, 2017). Situational method engineering was introduced generically as a process for constructing methods that are tuned to situation specificity, specialized by AMME for modelling methods regardless of their application and goals. For modelling languages and tools, other methodolo-gies and platforms aiming for flexible customization are also available (Kelly et al., 2013) (Frank, 2013) and could be included in the engineering method hereby proposed – i.e., if they are enriched with a knowledge streamlining approach to interoperate with RDF graph databases. This key ingredient was originally envisioned in (Karagiannis and Buchmann, 2016) and later deployed in several implementations (Buchmann and Karagiannis, 2015), (Buchmann and Karagiannis, 2016).

The proposed method generalizes and repurposes earlier attempts of applying reasoning on models (Corea and Delfmann, 2017). Workflow management systems, traditionally driven by XML serialization of standard process descriptions - e.g., XPDL (WfMC, 2017) - may also benefit from this proposal, if models represent processes and their execution context (as highlighted in the presented illustrative example). The Semantic Business Process Management paradigm (Hepp and Roman, 2007), which aims to splice Web Services, semantic technology and business process modelling also takes a knowledge-centric approach to processes. It emphasizes process checking and composition under ontological frameworks, showing less interest in their impact on agile software engineering methods.

Since the semantic parameterization of the developed software artefacts replaces the more traditional code generation practices, the traditional roundtrip engineering challenges (Maciaszek, 2002) must be reconsidered. An ADOxx plug-in ensures that the RDF graph's schema is kept in synch with metamodel changes for any modelling tool implemented on ADOxx (Open Models Laboratory, 2017). A roundtrip engineering cycle can be devised as a generalization of the proposed method– however, currently we only consider the benefits of the unidirectional knowledge flow from the model base back-end to the model-aware front-end.

6 CONCLUDING DISCUSSION

Figure 5 shows empirically observed efforts averaged over three development processes where the hereby proposed method was employed. The chart shows the evolution, across the 5 iterations, of several indicators corresponding to the knowledge conversion flow phases presented earlier in Figure 5:

For the *conceptualization effort* we have isolated only the microiteration cycle (language design and modelling tool implementation). We also subtracted the AMME training and the learning curve of nonexperienced modelling method engineers, as this develops a skill that is reused across projects. Once a modelling tool was prototyped, the *modelling effort* was also isolated, this time including the learning of the modelling language which is significantly high in the initial iteration, until users gain initial hands-on experience. The *data management effort* includes (i) the setup of the external data that does not make sense to be included in models (e.g., resource availability) and (ii) the setup of rules or OWL axioms to enrich the derived graphs according to the information needed in the front-end and (iii) the preparation of retrieval queries – e.g., SPARQL queries over HTTP and, in one case, a dereferencing mechanism for all model elements (Cinpoeru, 2017). Finally, the *model-driven implementation effort* covers the semantically parameterized implementation taking input from the model-data mashup and the inference results.

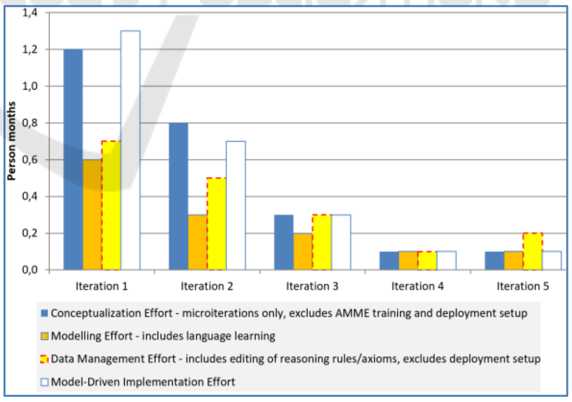


Figure 5: Empirically observed efforts across multiple implementations (averaged).

The chart shows a slow start due to the modelling language (re)implementation – however, this includes significant parts of the data model (including instance data, as shown in Figure 3) that otherwise would be created later; also, it relies on metamodelling platforms for fast prototyping. An essential benefit is that the setup is inherently prepared for integration

SCIENCE AND TECHNOLOGY PUBLICATIONS

within a Linked Data-based application environment. As this is a novel engineering approach that benefits from the interplay of Agile Modelling Method Engineering, semantic technology and model-driven software development, the existing experience is still limited and must be subjected to comparisons with traditional agile processes in order to quantify the trade-off between agile semantic richness and specificity in models and the benefits of standards-based code generators. Just as with the maturation of agile development practices, the uptake of the proposed *Model-Aware Software Engineering* approach depends on an accumulation of learned lessons from experimentation-oriented projects.

ACKNOWLEDGEMENTS

This work is supported by the Romanian National Research Authority through UEFISCDI, under grant agreement PN-III-P2-2.1-PED-2016-1140.

REFERENCES

Agile Manifesto, 2001. http://www.agilemanifesto.org, last accessed: 30th October 2017.

Ambler, S. W., 2002. *Agile Modeling: effective practices for Extreme Programming and the Unified Process*, Wiley.

BOC GmbH, ADOxx platform page – official website, 2017, http://www.adoxx.org/live, last accessed: 30th October 2017.

Buchmann, R. A., Karagiannis, D., 2015. *Agile Modelling Method Engineering: lessons learned in the ComVantage project.* In *Proceedings of PoEM 2015*, LNBIP 235, pp. 356-373, Springer.

Buchmann, R. A., Karagiannis, D., 2016. *Enriching Linked Data with semantics from domain-specific diagrammatic models*. In *Business and Information Systems Engineering*, 58(5), pp. 341-353, Springer.

Cinpoeru, M., 2017. *Dereferencing service for navigating enterprise knowledge structures from diagrammatic representations.* In *Proceedings of BIS 2017 Workshops,* LNBIP 303, pp.85-96, Springer.

Clark, T., Gonzalez-Perez, C., Henderson-Sellers, B., 2014. *A foundation for multi-level modelling*. In *Proceedings of Multi-level Modelling Workshop at MoDELS 2014*, CEUR-WS 1286, pp. 43-52.

Corcho, O., Fernandez-Lopez M., Gomez-Perez, A., 2003, *Methodologies, tools and languages for building ontologies. Where is their meeting point?* In *Data and Knowledge Engineering*, 46(1), pp. 41-64, Elsevier.

Corea, C., Delfmann, P., 2017. *Detecting compliance with business rules in ontology-based process modeling.* In *Wirtschaftsinformatik 2017 Proceedings*, pp. 226-240.

Frank, U., 2013. *Domain-specific modeling languages: requirements analysis and design guidelines*. In *Domain Engineering*, pp. 133–157, Springer.

Hepp, M., Roman, D., 2007. *An ontology framework for semantic business process management*. In *Wirtschaftsinformatik 2007 Proceedings*, 27.

Karagiannis, D., 2015. *Agile modeling method engineering*. In *Proceedings of the 19th Panhellenic Conf. on Informatics*, pp. 5-10, ACM.

Karagiannis, D., Buchmann, R. A., 2016. *Linked Open Models: extending Linked Open Data with conceptual model information.* In *Information Systems* 56, pp. 174-197, Elsevier.

Karagiannis, D., Buchmann, R. A., 2018. *A proposal for deploying hybrid knowledge bases: the ADOxx-to-GraphDB interoperability case*. In *Proceedings of the 51st HICSS 2018*, pp. 4055-4064, University of Hawaii.

Karagiannis, D., Kühn, H., 2002. *Metamodelling platforms*. In *Proceedings of the Third International Conference EC-Web 2002 – DEXA 2002*, Aix-en-Provence, France, LNCS 2455, pp. 182, Springer.

Karagiannis, D., Mayr, H., C., Mylopoulos, J. (eds.), 2016, *Domain-specific conceptual modelling*, Springer.

Kelly, S., Lyytinen, K., Rossi, M., 2013. *MetaEdit+ A fully configurable multi-user and multi-tool CASE and CAME environment*. In *Seminal Contributions to Information Systems Engineering,* pp. 109-129, Springer.

Kumar, K., Welke, R. J., 1992. *Methodology engineering: a proposal for situation-specific methodology construction.* In *Challenges and strategies for research in systems development,* pp. 257-269, Wiley & Sons.

Maciaszek, L. A., 2002. *Process model for round-trip engineering with relational database.* In *Successful Software Reengineering,* pp.76-91, IGI Global

Ontotext, 2017. GraphDB, http://graphdb.ontotext.com/, last accessed: 30th October 2017.

Open Models Laboratory, 2017, The EnterKnow project page, http://austria.omilab.org/psm/content/enterknow, last accessed: 30th October 2017.

van der Aalst, W. M. P., 2009. *Process-aware information systems: lessons to be learned from process mining.* In *Transactions on Petri Nets and Other Models of Concurrency II*, LNCS 5460, Springer, Berlin-Heidelberg, pp. 1-26.

W3C, 2017a, RDF - Semantic Web Standards. https://www.w3.org/RDF/, last accessed: 30th October 2017.

W3C, 2017b, SPARQL 1.1 Overview. https://www.w3.org/TR/sparql11-overview/, last accessed: 30th October 2017.

WfMC, 2017. XML Process Definition Language (XPDL), http://www.xpdl.org/, last accessed: 30th October, 2017.