# Agile Modelling Method Engineering: Lessons Learned in the ComVantage Research Project

Robert Andrei Buchmann[1(✉)] and Dimitris Karagiannis[2]

[1] Faculty of Economic Sciences and Business Administration, Babes-Bolyai University, Cluj-Napoca, Romania
robert.buchmann@econ.ubbcluj.ro
[2] Faculty of Computer Science, University of Vienna, Vienna, Austria
dk@dke.univie.ac.at

**Abstract.** The paper reports on experiences accumulated during a EU research project where challenges pertaining to requirements-driven metamodelling agility have been analysed. Traditionally, modelling languages are perceived as stable artefacts – that is, if they address a sufficiently large community with fixed modelling requirements on a fixed layer of abstraction. However, the enterprise modelling community must also consider the case where evolving requirements emerge in a narrow domain, or even in a single enterprise, therefore reusability across domains will be sacrificed to the benefit of on-demand adaptation, specialization or integration. Under such conditions, an agile metamodelling approach was applied in the ComVantage project and this, in turn, raised specific requirements for conceptual and technological enablers, allowing us to derive conclusions that are generalized here beyond the project scope. The paper's concluding SWOT analysis highlights the need to stimulate the emergence of an agile metamodelling paradigm based on community-driven enablers.

**Keywords:** Agile modelling method engineering · Enterprise modelling · Metamodelling · Modelling requirements

## 1 Introduction

Diagrammatic conceptual modelling has been concerned with standardization even from its earliest days, when the first draft of "process chart" symbols [1] was presented to the American Society of Mechanical Engineers (ASME), with a subtitle suggesting optimality: "Process Charts – First Steps in Finding the One Best Way to Do Work". According to the authors, the process charts were "a device for visualizing a process as means for improving it" and, for this purpose, an initial set of quite arbitrarily chosen symbols was proposed. The set of symbols was further developed by ASME in a new set of flowcharting symbols whose key quality was that they could be drawn by engineers using crayon and "template rulers". Later, Von Neumann adopted these to describe the first programs in a control flow visualization style. Many years later, standards like UML and BPMN still inherit the rhombus shape for decision concepts


J. Ralyté et al. (Eds.): PoEM 2015, LNBIP 235, pp. 356–373, 2015.
DOI: 10.1007/978-3-319-25897-3_23

(and other such "legacy symbols") as part of standardized notations. However, computer-aided diagrammatic modelling can now benefit from enriching notation with a variety of features, such as: *interactivity* (symbols acting as hyperlinks), *dynamics* (symbols changing based on some machine-readable properties or semantics), *visual semantics* (information communicated through ornamental aspects, or even animation). None of these had been possible in the days of crayon and template rulers, and the variety of available options in this respect is nowadays subject to **modelling requirements**, together with other customization needs of the targeted users (modellers) pertaining to (i) model-driven functionality, (ii) semantic coverage of models or (iii) the depth of domain-specificity.

As software engineering recognized the unstable nature of requirements and the competitive value of flexible response to evolving requirements, the Agile Manifesto [2] and community emerged to challenge the traditional rigid ways of building software. Similar challenges can be met with respect to modelling requirements in the practice of modelling method engineering, particularly in its application for enterprise modelling. Consequently, an Agile Modelling Method Engineering (AMME) approach and its enablers must be consolidated from experience reports such as the one to be provided in the work at hand.

The goal of this paper is to share metamodelling experiences from our research on modelling virtual enterprises in the ComVantage research project [3], consequently highlighting characteristics of AMME as a key methodological necessity. In order to establish the motivation, we will introduce the improvised term "ComVantage enterprise" and compile its characteristics and challenges that must be tackled with an AMME approach. A modelling method and prototype was developed for the mentioned project within the environment provided by the Open Model Initiative Laboratory [4], and the paper reflects back on this development experience.

The remainder of the paper is structured as follows: Sect. 2 provides a brief project overview and, as a generalized motivation, compiles the enterprise characteristics that require an AMME approach. Section 3 provides an overview on the emerging AMME approach and its key enablers. Section 4 reports on experiences with AMME in relation to examples extracted from the ComVantage modelling method. Section 5 positions the paper in the context of related works. The final section generalizes conclusions beyond the project scope.

## 2 Motivational Characteristics of a "ComVantage Enterprise"

In this section we will compile characteristics for a "ComVantage enterprise" from the project experience, as well as by instantiating features identified in research roadmaps provided by European research clusters where the project took part - e.g., *FInES* [5].

The ComVantage research project aimed to support collaborative business processes in virtual enterprises with adaptive mobile technology driven by models designed with the domain-specific "ComVantage modelling method". The modelling method had to be developed iteratively, as requirements evolved and the development of run-time systems also imposed an incremental iterative approach. Details on

different snapshots of the evolving modelling method are available in deliverables [3] (deliverables D311, D312) and previous publications [6, 7]. In this paper we will only provide insight for a few modelling elements in order to derive key characteristics for AMME. In the following, we highlight the enterprise characteristics that must be tackled by the AMME approach:

**A. Modelling is Employed as Knowledge Representation.** Historically, model-driven software engineering employed models in different ways, as highlighted by Table 1. The "modelling is programming" paradigm [8] uses models for generation of executable code. This vision was later complemented by a "modelling is configuration" approach, having run-time systems parameterized with model information and models acting as "control panels" to influence run-time behaviour – e.g., in the context of process-aware information systems [9], with the help of XML model serializations such as BPEL [10]). As enterprise modelling expanded beyond the goals of business process management in order to build a holistic representation of the enterprise, diagrammatic models became means of capturing the knowledge of stakeholders on different enterprise aspects (e.g. processes, goals, capabilities).

**Table 1.** The different stages of model-driven software engineering

| How models are employed | Relation to run-time systems | Pre-condition |
|---|---|---|
| *Modelling is Programming* | Run-time code is generated from model | All information necessary in the final code is available, explicitly or implicitly, in models |
| *Modelling is Configuration* | Run-time system is parameterized and its execution driven by model information | Model information can be serialized in some interoperable format, typically XML |
| *Modelling is Knowledge representation* | Run-time system functionality is influenced by properties, concepts and reasoning based on the semantics captured in models | Model information can be converted to some machine-readable knowledge base (that supports querying and reasoning) |

**Relevance to AMME:** AMME is called to ensure that the necessary semantics is captured, in relation to modelling needs that may evolve as users become accustomed to modelling. On the side of stakeholders, inexperienced modellers will gradually raise requirements for deeper specialization of modelling concepts (deepening domain-specificity), while the modelling method engineer can be confronted with gradual understanding of the application domain.

**B. Semantics-Awareness is Required for Run-Time Enterprise Systems.** The functionality of enterprise systems can be, at execution time, sensitive to machine-readable semantics captured in various forms. Typically such semantic representations are nowadays available in a new type of semantic networks (the Linked Enterprise Data paradigm [11]), possibly enriched by ontologies and rules to enable reasoning over the

Web of Data. Alternatively, *information systems may also be sensitive to the semantics captured in diagrammatic form by enterprise models* (with the modelling language alphabet acting as a "terminological box") – that is, if relevant granularity is ensured, and a machine-readable model repository is provided.

**Relevance to AMME:** AMME is called to ensure that changes in requirements for the run-time enterprise system propagate accordingly to modelling requirements so that the necessary properties are assimilated in the modelling language (to become accessible at run-time).

**C. Scenario-Driven Requirements for Run-Time Systems.** Means that run-time requirements should be built around work processes and work capabilities rather than around disparate use cases. While traditional requirements elicitation practices used to advocate atomization of user stories, the increasing experience of stakeholders with business process management may be leveraged in order to capture requirements in control flow representations mapped on resource/capability requirements. Agile software engineering already recognized this by aggregating requirements in narrative units such as stories and epics.

**Relevance to AMME:** As [12] emphasizes, requirements elicitation does not produce requirements per se, but requirements representations, whose descriptions may involve domain-specific concepts and properties relative to the business context. AMME is called to provide agile modelling means for collecting enterprise system requirements that include domain-specific aspects, thus bridging the gap between stakeholders and system developers [7].

**D. Complexity Management Challenges.** Although modelling typically means abstraction and simplification, in enterprise modelling even the model is too complex to be comprehensive in a single diagrammatic representation. Enterprise models are partitioned in different facets/layers across different abstraction layers or viewpoints (see Archimate [13], Zachman [14]). Since diagrammatic enterprise models are subjected to human interpretation, an inherent decomposition requirement must ensure model understandability and should be satisfied both on the *meta level* (e.g., by partitioning a metamodel in multiple model types) and on the *model level* (e.g., decomposing a business process model in subprocesses).

**Relevance to AMME:** AMME is called to identify building blocks of manageable granularity for both modellers (to help with model comprehension) and metamodellers (to help defining backlog units). Such building blocks will make it possible to isolate evolving modelling requirements so that an agile response to changes does not affect the entire method and change propagations become traceable.

**E. Domain-Specific Modelling Requirements.** Enterprise models can be specialized and contextual, and it is plausible to have an enterprise modelling method adopted in a narrow community, or even for a single case/project, where the enterprise is interested in employing models for internal purposes (e.g., as input for a custom model-aware run-time system), with no desire for sharing and reusing them in the external environment.

**Relevance to AMME:** In domain-specific modelling, the level of concept specialization is not necessarily fixed, as new properties and subtyping may be gradually required to achieve new capabilities and the proverbial "competitive advantage" of agile development. AMME is called to ensure that such evolving specialization is assimilated in the method in a timely manner.

## 3 The AMME Framework

### 3.1 Framework Overview

Without going point by point through the Agile Manifesto [2], we highlight the key characteristics of Agile development as compiled by [15]:

- *Iterative*: repeat activities and potentially revisit same work products;
- *Incremental*: each successive version is usable and builds upon previous versions;
- *Version control*: enable for other agile practices;
- *Team control*: small group of people assigned to the same project building block with shared accountability.

In order to achieve agility in Modelling Method Engineering, such principles must be grafted on the fundamentals of modelling method design, considering the building blocks defined in [16], described as follows:

(1) The *modelling language* describes the set of modelling constructs (their notation, grammar and semantics. To achieve manageable granularity and model comprehensibility, the modelling language can be partitioned in model types addressing different facets or abstraction layers of the system under study. The partitioning can be perceived as a usability feature (a top-down decomposition approach to avoid visual cluttering) or a consequence of hybridization (a bottom-up strategy employed to interconnect modelling language fragments).
(2) The *modelling procedure* defines the steps that must be taken by modellers towards some goal (in the simplest case, it advises on the precedence in creating different types of models).
(3) The *mechanisms and algorithms* cover functionality that process model information for various purposes (visualization, transformation, evaluation etc.).

Agility relies on *iterative incremental cycles integrated in a high level framework* such as the one depicted in Fig. 1.

In the centre, the modelling method evolves in a *Produce-Use* metamodelling cycle (by analogy with the *Code-Test* cycle): (i) in the top phase, the modelling language is derived from so called "Models of Concepts" that capture the domain knowledge and structure it in modelling constructs (the language alphabet); (ii) in the lower phase, each iteration of the modelling method is implemented in a modelling tool that allows the creation and evaluation of enterprise models ("Models that Use Concepts") by instantiating the concepts designed in the previous phase/For this cycle, the Application Environment raises modelling requirements and provides domain knowledge, while in the backend a Knowledge and Resource Repository accumulates reusable resources

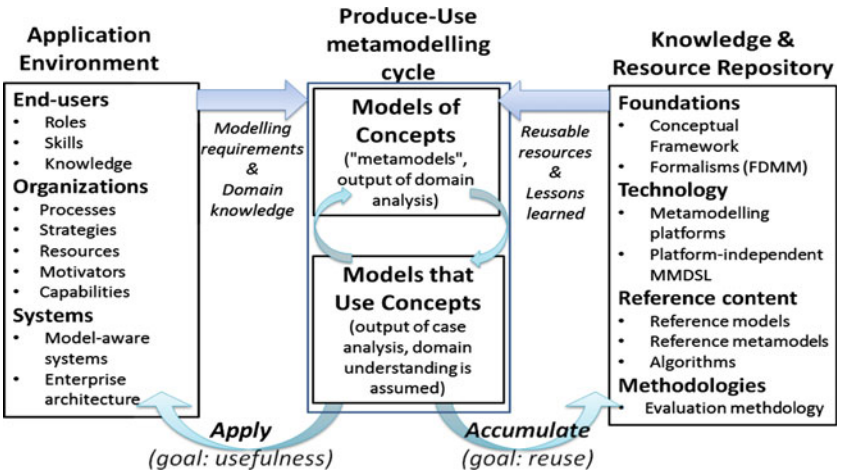


**Fig. 1.** The AMME Framework (adapted from [17])

and lessons learned. In our case the Knowledge and Resource Repository is accumulated through the Open Model Initiative Laboratory, a physical and virtual research environment that also hosts an implementation of the ComVantage modelling prototype [4] (which still evolves in follow-up projects).

### 3.2 Conceptual Enablers

During the *Produce* phase of the cycle depicted in Fig. 1, a knowledge acquisition effort will produce a "Model of Concepts" to describe the enterprise ontologically. A practical requirement for decomposition naturally emerges in order to ensure model comprehensibility and a usable separation of concerns. The solutions that fulfil the decomposition requirement are also the basis of establishing a manageable backlog granularity during AMME, beyond the building blocks suggested in Sect. 3.1. A generic classification scheme for different kinds of units (content containers) is provided in Fig. 2.

**On the Top Layer**: *Asemantic containers* provide a grouping of modelling constructs without assigning explicit machine-readable semantics to this grouping, unlike the *semantic containers* which have to their content a richer relation than the generic mereological one ("part-of"/"contains"). Depending on how containers are perceived, we may have *visual containers* (visual partitions sectioning the modelling canvas according to some criteria, e.g. business process swimlanes) or *functional containers* (an entire model). With respect to how the contents of a container are related to the exterior of that container, we may have *related content* (through machine-readable relations) or isolated content.

**On the Lower Layer:** The container types are further subsumed to more specialized building blocks: for example *swimlanes/pools* in business process models are typically

*unspecified visual containers* (their meaning can be left to human interpretation). However, depending on the freedom of interpretation to be allowed, the relation of a swimlane to its contents may also be prescribed at metamodel level, by imposing semantics on this containment (e.g., a machine-readable link to an organisational unit). A more complex type of container is the *model type*, used when the enterprise modelling language has the alphabet split into problem-specific subsets. These are *semantic containers* in the sense that they have a clear meta-level specification of what concepts are allowed in each type of model, as well as constrained relations (links) to other models. If such a specification does not exist, we are talking about *unprescribed canvas* (e.g. Powerpoint-style free sketches). Each concept in a model type becomes a semantic unit, and its properties must be traced to detect propagation of changing requirements, as will be shown in Sect. 4. Further (implementation-specific) specialization is suggested in Fig. 2, however it will not be detailed here since it depends on the intended depth of domain-specificity on a case by case basis.

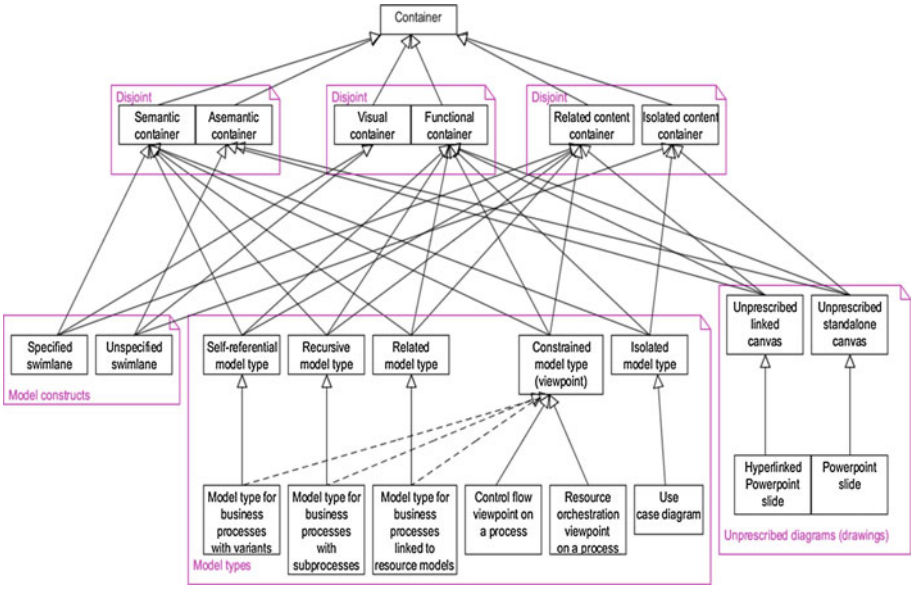


**Fig. 2.** A taxonomy of modelling units to tackle the decomposition requirement

### 3.3 Technological Enablers

Rapid prototyping is a key necessity for producing iterative prototypes that meet gradually refined requirements, even "throwaway prototypes" aimed to (i) familiarize inexperienced modellers with what they can expect from the final implementation and to (ii) stimulate their ability to formulate modelling requirements. The practice of metamodelling has answered such necessities and nowadays we see an increasing popularity of platforms that provide built-in functionality for productive development of modelling tools according to some existing conceptualisation (e.g., ADOxx [18],

MetaEdit+ [19]). Each available metamodelling platforms relies on a meta$^2$model providing primitive built-in constructs (e.g., concept, connector) to create "Models of Concepts". As a complement to the available metamodelling platforms, specifically aimed to support rapid cross-platform prototyping, we have proposed in [20] an *additional abstraction layer* where a modelling method can be described in a declarative language (**MM-DSL**, Fig. 3) built on a common subset of primitives of the popular meta$^2$models. A draft of the language grammar is openly available at [21]. Rapid prototyping is enabled by compilers that translate MM-DSL code to a platform-specific format for the metamodelling platform of choice in order to produce modelling prototypes in a highly automated way (a proof-of-concept compiler is currently available only for ADOxx deployment, with additional compilers expected to emerge from the OMILab community).

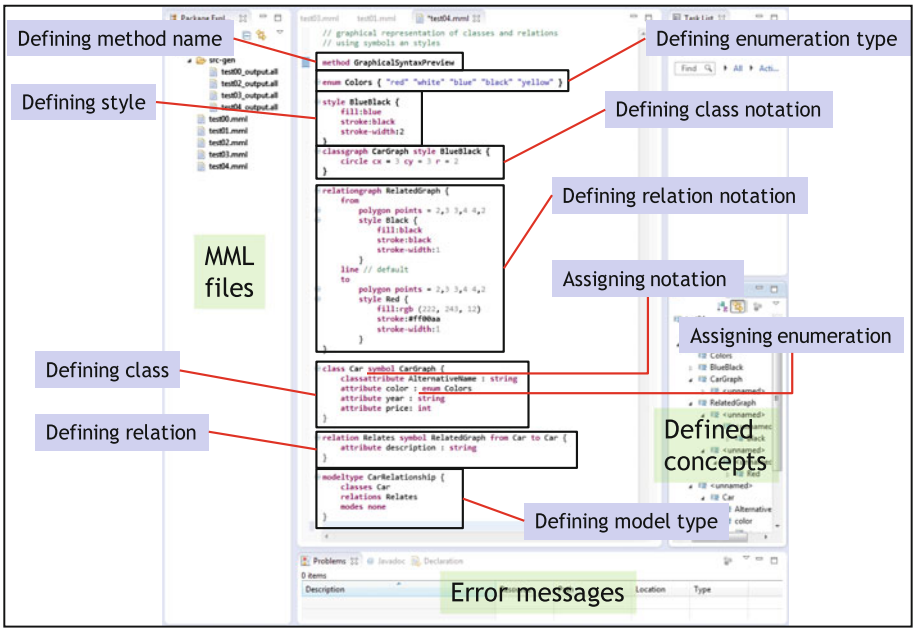


**Fig. 3.** The blocks of modelling method definition with MM-DSL (details on code snippets and grammar available in [20, 21])

## 4 AMME: The ComVantage Case

Enterprise modelling methods typically show a recurring frame of facets/layers/viewpoints and this was also adopted in the ComVantage modelling method, as its metamodel is split in various enterprise facets – the business model, business processes, app and data requirements (due to the project's technological specificity), models of domain-specific resources. Semantic relations are present in the form of hyperlinks that allow navigability across models and at the same time characterize the semantics of

how the different facets (typically covered by different model types) are related one to another. We will not discuss here the entire metamodel layering (details available in public deliverables [3, D312]), only some fragments that illustrate the modelling requirements evolution and its propagation in the method building blocks.

Since decomposition is a key enabler for agility, the modelling method building blocks *must be mapped on backlog tasks and sprint responsibilities*. Metamodellers are typically confronted with requirements that propagate across these building blocks, with typical situations being reflected in Fig. 4 according to the following numbering (concrete project-based examples follow in the subsequent subsection):

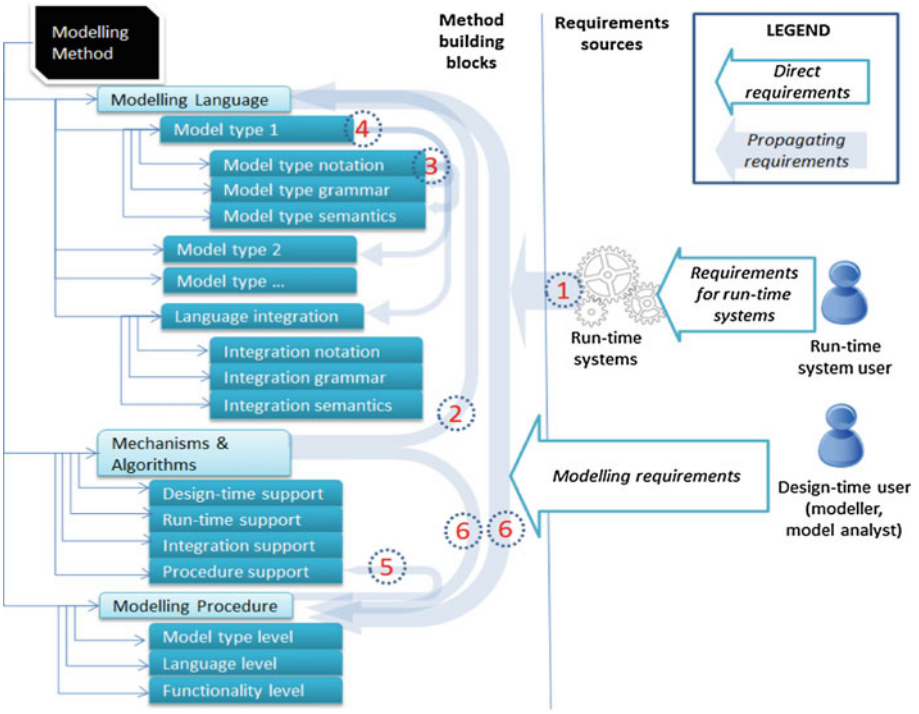


**Fig. 4.** Propagation of modelling requirements across method building blocks

1. Requirements for (semantics-aware) run-time systems will propagate in requirements for semantics available in models;
2. Requirements on mechanisms and algorithms typically propagate in requirements for new properties or concepts in the language (to provide additional input to functionality);
3. Requirements on notation change quite often, as most users perceive modelling primarily on a visual level. Change requests on notation level typically propagate in the other building blocks, e.g.: (i) dynamic or interactive notations will depend on

the presence of some property in the concept semantics; (ii) excessive visual cluttering will be solved by splitting a model type in multiple model types, also raising new integration needs (hyperlinks between models);
4. Requirements on a model type typically propagate in other related model types, as concepts are transferred among model types to reduce the linking effort, or new concepts are introduced;
5. The modelling procedure typically takes the form of modelling guidelines, therefore usually there are no direct requirements on the modelling procedure. However, users may impose requirements in the form of constraints, which are quite volatile in nature (e.g., "I don't want to have more than 5 model types"). More importantly, such constraints refer to usability and the automation of some procedure steps (e.g., "I don't want to create models of type X. Instead, they should be generated automatically") and this typically propagates towards additional functionality;
6. Any changes in the modelling language and functionality propagate in the modelling procedure guidelines.

A core concept of the method is the **App requirement** concept aiming to capture, at business process level, what kinds of mobile apps are required to support specific business activities. As an app-centred run-time architecture was being developed in the project, the modelling requirements on how the "required apps" should be described evolved. Table 2 illustrates this evolution along 5 phases, while the subsequent figures reflect how the concept evolved on a diagrammatic level.

**Table 2.** Evolving "required App" concept in relation to modelling requirements

| | Modelling requirement | Solution | Propagations to backlog items |
|---|---|---|---|
| Phase 1 | The business process activities should indicate when an activity must be supported by a mobile app | The concept of business process Activity gets some editable properties where the modeller (e.g., app requirements elicitator) captures descriptions of the app capabilities required to support that activity | The concept of Activity in the business process model type is extended with new properties. The modelling procedure guidelines must indicate how the app requirements are expressed (see a mockup in Fig. 5) |
| Phase 2 | The mobile app requirement should be symbolised by its own concept with domain-specific properties to describe the required app | All semantics pertaining to the app requirements are isolated in a new app concept. Apps are visually connected to activities to indicate where a mobile app is required. Each app | The concept of Activity loses some properties. The App concept is added to the language alphabet (i.e., to the business process model type), with new editable properties. The modelling |

(*Continued*)

**Table 2.** (*Continued*)

| | Modelling requirement | Solution | Propagations to backlog items |
|---|---|---|---|
| | | symbol is further described by its own property sheet capturing various non-functional requirements (e.g., device type, operating system) | procedure guidelines must reflect the new way of connecting app requirements to activities |
| Phase 3 | To avoid visual cluttering, the mobile app symbols should be linked outside the business process modelling canvas and collected in a pool of reusable app requirements | A new model type is created to collect the app symbols in a single catalogue of "required apps". The notation in the business process activity provides a hyperlink to easily navigate to the app description linked to each activity. This is visible in Fig. 6 as an app icon in the top left corner of the app-supported activities | The business process model type is split, so that a new model type will include only the app symbols. The connector between activities and apps is replaced by a hyperlink between the two model types (on the Activity notation). The modelling procedure guidelines must be updated with the new way of linking app requirements, as well as with the prerequisite of having an app element available before a hyperlink is created for it |
| Phase 4 | Each required mobile app should be further described by its features in the form of abstract user interaction elements of different types (e.g., buttons, labels etc.). This requirement aims to provide a basis for early mockup designs for the required app, thus making the elicitation of app requirements "bleed | A new model type is added to describe an abstract user interface (similar to the "abstract UI" concept in the Cameleon framework [22]). This expresses the key app features that must be available for user interaction in each required app | The metamodel is extended with a new model type, describing an app user interface in terms of some abstract types of UI controls (a taxonomy of such controls must be devised). A hyperlink is enabled between app symbols and such UI models. The modelling procedure guidelines must |

(*Continued*)

**Table 2.** (*Continued*)

| | Modelling requirement | Solution | Propagations to backlog items |
|---|---|---|---|
| | into" the app design phase | | include explanations on the proposed taxonomy of UI controls and how the new type of model should be linked |
| Phase 5 | An external app orchestration system will automatically deploy and execute mobile apps that are chained according to the workflow dictated by the business process model. The orchestration engine should be model-aware in the sense that the precedence of app chaining must be dictated by the precedence of the apps modelled for the business process (details in [23]) | (a) A new model type depicts the app precedence;<br>(b) Functionality for automated derivation of the app precedence from existing business process models;<br>(c) Functionality for exporting the resulted diagrams in a serialization format that can be queried by the app orchestration engine responsible with app deployment (RDF [24] was the format of choice) | The metamodel is extended with a new model type (the "orchestration", describing the precedence of app usage for a business process. A mechanism is needed to derive such models from business process models, in order to ensure their consistency. Another mechanism is created to serialize the contents of models in some query-able format, to facilitate the extraction of semantics from the model information). Modelling procedure guidelines must be updated to (a) instruct the user in using these mechanisms;<br>(b) instruct the run-time system developer on the RDF vocabulary that must be used to build model queries |

Figure 5 shows how the app requirement description evolved between Phases 1 and 2, from an editable property of a process activity to a modelling symbol with its own domain-specific editable properties (the blue boxes are mock-ups of the property sheets).

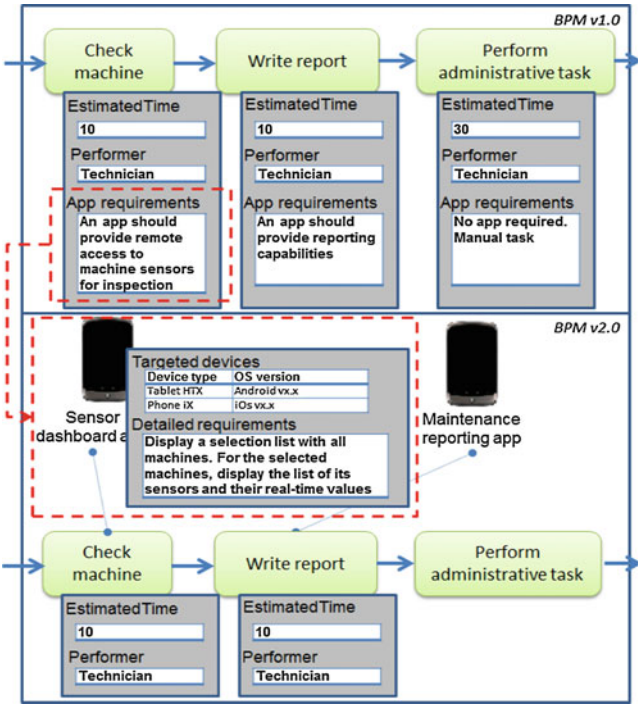


**Fig. 5.** Descriptive property evolves into modelling concept (Phase 1–2 requirements) (Color figure online)

Figure 6 shows the additional model types that emerged from Phases 3 and 4 (the pool of required apps isolated from the business process model, and the abstract UI model for each app), as well as the types of links that can be created between models (see the legend). Figure 7 shows the orchestration model type emerging in Phase 5 as a necessity for exposing the app usage precedence to the run-time system (run-time orchestrated apps [23]). The sample model in Fig. 7 is generated automatically via a graph rewriting rule set applied on the business process model example in Fig. 6 (details on the transformation are out of this paper's scope, being available in [7]).

## 5 Related Works

To the best of our knowledge, the framework of Agile Modelling Method Engineering was initially outlined on a generic level in some of our previous work - the published keynote [17] and the works of the NEMO (*Next Generation Enterprise Modelling)* Summer School [25], with some characteristics being suggested (in the context of fast prototyping) in [20]. This paper is distinguished from the mentioned works by (a) reporting on experiences from an application case of the generic AMME framework, with respect to challenges identified in the ComVantage research project (the case

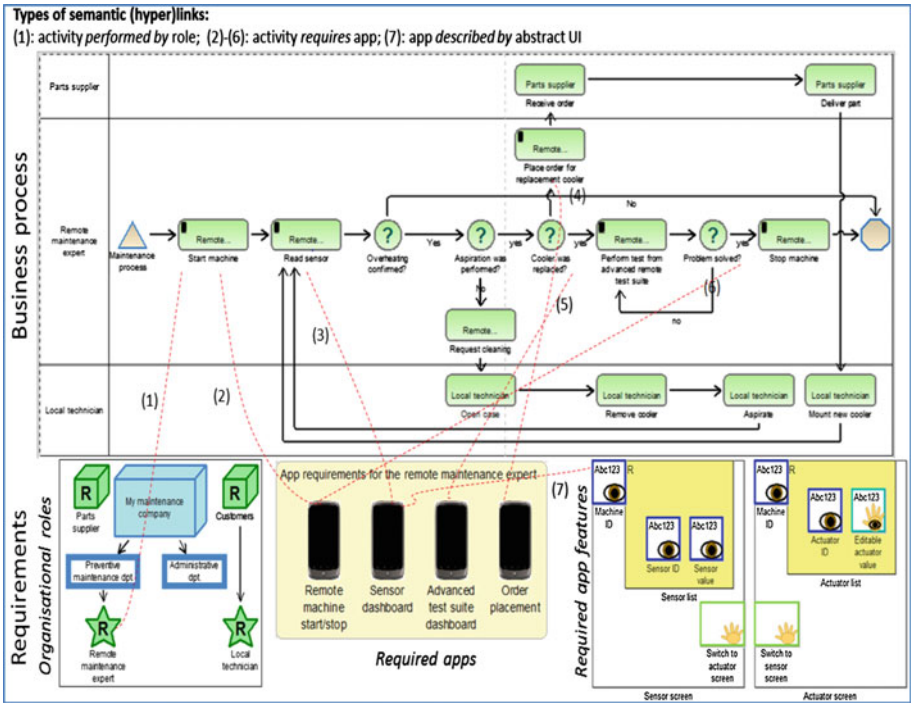


**Fig. 6.** Modelling concept evolves into linked model type (Phase 3–4 requirements)

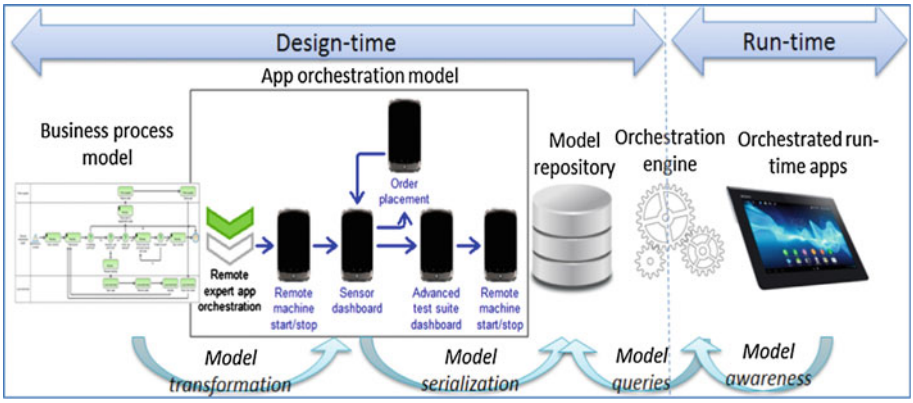


**Fig. 7.** Linked model types enriched with functionality (Phase 5 requirements)

discussed in Sect. 4); (b) refining the characteristics of a "Next Generation Enterprise" by anchoring them in characteristics of enterprises involved in the discussed project (Sect. 2); and (c) providing details on the conceptual and technological enablers (Sects. 3.2 and 3.3).

However, the AMME framework builds on agility challenges identified in other related works, since it emerged from a multi-disciplinary convergence of concerns previously discussed in the fields of Metamodelling and Enterprise Modelling:

Agility challenges, as well as model complexity challenges, have been discussed in the field of Metamodelling with respect to improving productivity for multi-perspective modelling [26], domain-specific multi-level modelling [27], as well as in a meta-modelling interpretation on Language-oriented Programming [28]. Metamodelling itself, as a discipline, emerged from the need to enable flexibility in modelling language design through a multi-layered abstraction architecture (e.g., the "powertype" approach [29], the MetaEdit+ approach [30]). This, however, has not been complemented yet by a fully-fledged methodological agile approach that mirrors the agility principles and challenges from software development. Project-based experiences are necessary to bring forth the kind of reflections that fuelled the agile movement in software engineering, since the Agile Manifesto mainly evolved from pragmatic needs of practitioners who questioned the obstacles and pitfalls of rigid management approaches (e.g., the waterfall approach).

The discipline of Enterprise Modelling recognizes a need for semantic as well as notational diversity, since a multitude of enterprise architectures and frameworks have been established and are widely adopted (e.g., EKD [31], ArchiMate [13], Zachman [14], more recently capability-driven approaches [32]). However, the paradigm of Enterprise Modelling traditionally obscures the notion of **evolving modelling requirements** which is central to AMME, as it requires dedicated management strategies. This is perhaps due to ambitions for standardization, which is in line with the original goal of ASME process charts on enabling "the best way to do work". The work at hand aims to stimulate the assimilation of "unstable" domain-specificity (or even case-specificity) that must extend high-level enterprise architecture models in response to changing requirements.

## 6 Concluding SWOT Analysis

Just like the Agile Manifesto for software development, the notion of AMME is consolidated not only from theoretical analysis, but also from reflecting back on obstacles and experiences with metamodelling projects. Experiences with the ComVantage research project show a practical AMME instantiation and have been synthesized in the work at hand. Consequently, AMME is hereby analysed qualitatively as a foundational notion with an early-stage maturity level on which future work is called to further establish enablers. By paralleling the principles of Agile software development, methodological or technological enablers are expected to emerge from community-driven efforts and shared experiences: management support tools, rapid prototyping enablers, validation methodologies, reports on agile best practices or agility pitfalls. Since we are not dealing yet with a wide adoption of AMME, it is too early to assess comparative experience reports. A SWOT analysis is hereby provided to evaluate the success of AMME for the instance experience:

**Strengths**: AMME was a necessity for enabling the evolution of Enterprise Modelling requirements and for dealing with their multi-faceted nature, as multiple types of stakeholders have been involved, from design-time decision makers (aiming for model analysis) to run-time system users (relying on machine-readable model semantics). Conceptual and technological enablers supported the agile approach and brought forward the need for a mature agile approach to metamodeling.

**Weaknesses**: The experience with applying AMME is still limited to OMILab projects. Its application to ComVantage emerged as a necessity driven by unstable requirements. The main identified pitfalls are (a) that synchronicity with the development of the run-time systems relying on model information is difficult to maintain if not planned from the very beginning; and (b) an evaluation methodology for user acceptance involves a training phase that can create bottlenecks, as stakeholders are called to frequently re-learn the modelling language. In addition, aspects pertaining to the management of human resources in agile teams have not been tackled in this paper. Future works must consider strategies to tackle such challenges.

**Opportunities**: The evolution of the Agile software development practices and tool support was fundamentally community-driven. Future work may be layered on the conceptual foundation established by the work at hand in order to enrich AMME as a community-driven framework, to raise its maturity level, to enrich tool support and to enable longitudinal studies for further refinement.

**Threats**: Standards typically defuse the problems raised in the work at hand by bringing all potential users on the same level of abstraction and encouraging universal adoption for the benefit of reusability across domains. Therefore, the generalized relevance of the work at hand depends on the desired trade-off between reusability and domain specialization, as well as on the uptake of semantics-aware information systems whose evolving requirements are an inherent cause for propagating modelling requirements.

**Acknowledgment.** The research leading to these results was funded by the European Community's Seventh Framework Programme under grant agreement no. FP7-284928 ComVantage.

## References

1. Gilbreth, F.B., Gilbreth, L.M.: Process Charts. American Society of Mechanical Engineers (1921)
2. Manifesto for Agile Software Development. http://agilemanifesto.org/
3. ComVantage Consortium, ComVantage public deliverables. http://www.comvantage.eu/results-publications/public-deriverables/
4. Open Model Initiative Laboratory, ComVantage modelling prototype and resources. http://www.omilab.org/web/comvantage/home
5. Future Internet Enterprise Systems cluster, The FInES Research Roadmap 2025. http://cordis.europa.eu/fp7/ict/enet/documents/fines-research-roadmap-v30_en.pdf

6. Buchmann, R.: Conceptual modeling for mobile maintenance: the ComVantage case. In: Sprague, R.H. Jr. (ed.) Proceedings of HICSS 47, pp. 3390–3399. IEEE (2014)
7. Buchmann, R., Karagiannis, D.: Modelling mobile app requirements for semantic traceability. J. Requirements Eng. (2015, in press). doi:10.1007/s00766-015-0235-1
8. Aquino, N., Vanderdonckt, J., Panach, J.I., Pastor, O.: Conceptual modelling of interaction. In: Embley, D., Thalheim, B. (eds.) Handbook of Conceptual Modeling: Theory, Practice and Research Challenges, pp. 335–355. Springer, Berlin (2011)
9. van der Aalst, W.M.P.: Process-aware information systems: lessons to be learned from process mining. In: Jensen, K., van der Aalst, W.M.P. (eds.) Transactions on Petri Nets and Other Models of Concurrency II. LNCS, vol. 5460, pp. 1–26. Springer, Heidelberg (2009)
10. OASIS, BPEL - the official website. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
11. Wood, D. (ed.): Linking Enterprise Data. Springer, Berlin (2010)
12. Kaindl, H., Svetinovic, D.: On confusion between requirements and their representations. Requirements Eng. **15**, 307–311 (2010)
13. The Open Group, ArchiMate® 2.1 Specification. http://www.opengroup.org/archimate/
14. Zachman, J.A.: A framework for information systems architecture. IBM Syst. J. **26**(3), 276–292 (1987)
15. Agile Alliance. http://guide.agilealliance.org/subway.html
16. Karagiannis, D., Kühn, H.: Metamodelling platforms. In: Bauknecht, K., Min Tjoa, A., Quirchmayr, G. (eds.) EC-Web 2002. LNCS, vol. 2455, p. 182. Springer, Heidelberg (2002)
17. Karagiannis, D.: Agile modelling method engineering. In: Proceedings of the 19th Panhellenic Conference on Informatics. ACM (2015)
18. BOC-Group, ADOxx tool page. http://www.adoxx.org/live/
19. MetaCase, MetaEdit+ tool page. http://www.metacase.com/products.html
20. Visic, N., Fill, H.-G., Buchmann, R., Karagiannis, D.: A domain-specific language for modelling method definition: from requirements to grammar. In: Rolland, C., Anagnostopoulos, D., Loucopoulos, P., Gonzalez-Perez, C. (eds.) Proceedings of RCIS 2015, pp. 286–297. IEEE (2015)
21. The MM-DSL grammar specification. http://www.omilab.org/c/document_library/get_file?uuid=eb040aac-ea0d-4df7-a0a9-80b73f00c5f8&groupId=10122
22. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A unifying reference framework for multi-target user interfaces. Interact. Comput. **15**(3), 289–308 (2003)
23. Ziegler, J., Graube, M., Pfeffer, J., Urbas, L.: Beyond app-chaining - mobile app orchestration for efficient model driven software generation. In: Proceedings of EFTA 2012, pp. 1–8. IEEE (2012)
24. W3C, RDF 1.1 Concepts and Abstract Syntax. http://www.w3.org/TR/rdf11-concepts/
25. Open Model Initiative Laboratory, NEMO 2015 Summer School materials. http://www.omilab.org/web/guest/camp2015/topics-and-program
26. Frank, U.: Multi-perspective enterprise modeling: conceptual framework and modeling languages. In: Sprague, R.H. Jr. (ed.) Proceedings of HICSS 2002, pp. 72–82. IEEE (2002)
27. Frank, U.: Multilevel modeling: toward a new paradigm of conceptual modeling and information systems design. Bus. Inf. Syst. Eng. **6**(6), 319–337 (2014)
28. Clark, T., Sammut, P., Willans, J.: Applied metamodelling: a foundation for language driven development. http://eprints.mdx.ac.uk/6060/
29. Gonzalez-Perez, C., Henderson-Sellers, B.: Metamodelling for Software Engineering. Wiley, London (2008)

30. Kelly, S., Lyytinen, K., Rossi, M.: MetaEdit+ a fully configurable multi-user and multi-tool CASE and CAME environment. In: Bubenko, J., Krogstie, J., Pastor, O., Pernici, B., Rolland, C., Solvberg, A. (eds.) Seminal Contributions to Information Systems Engineering, pp. 109–129. Springer, Berlin (2013)
31. Loucopoulos, P., Kavakli, V.: Enterprise knowledge management and conceptual modelling. In: Chen, P.P., Akoka, J., Kangassalu, H., Thalheim, B. (eds.) Conceptual Modeling. LNCS, vol. 1565, pp. 123–143. Springer, Heidelberg (1999)
32. Zdravkovic, J., Stirna, J., Kuhr, J.-C., Koç, H.: Requirements engineering for capability driven development. In: Frank, U., Loucopoulos, P., Pastor, Ó., Petrounias, I. (eds.) PoEM 2014. LNBIP, vol. 197, pp. 193–207. Springer, Heidelberg (2014)