universität wien

Faculty of Computer Science

A Metamodeling Approach to Support the Engineering of Modeling Method Requirements

Dimitris Karagiannis and Patrik Burzynski and
Wilfrid Utz and Robert Andrei Buchmann

Published in:

2019 IEEE 27th International Requirements Engineering Conference (RE), Jeju Island, Korea (South), 2019, pp. 199-210.

Final version available at: https://doi.org/10.1109/RE.2019.00030

OMiLAB®

www.omilab.org

A Metamodeling Approach to Support the Engineering of Modeling Method Requirements

Dimitris Karagiannis, Patrik Burzynski, Wilfrid Utz
Research Group Knowledge Engineering
University of Vienna, Faculty of Computer Science
Vienna, Austria
{dk,patrik.burzynski,wilfrid}@univie.ac.at

Robert Andrei Buchmann
Business Informatics Research Center,
University Babeș-Bolyai
Cluj Napoca, Romania
robert.buchmann@econ.ubbcluj.ro

***Abstract* — The notion of "modeling method requirements" refers to a category typically neglected by RE taxonomies and frameworks – i.e., those requirements that motivate the realization of (conceptual) modeling methods and tools. They can be considered domain-specific, in the sense that all modeling methods provide a knowledge schema for some selected application domain (narrow or broad). Besides this inherent domain-specific nature, we are investigating how the characteristics of modeling methods inform the RE perspective, and how in turn RE can support the engineering of such artifacts.**

Thus, the work at hand aims to raise awareness about modeling method requirements in the RE community. The core contribution is the CoChaCo (Concept-Characteristic-Connector) method for the representation and management of such requirements, as well as for streamlining with subsequent engineering phases. CoChaCo is itself a modeling method – i.e., it achieves its goals through diagrammatic modeling means for which a supporting tool was prototyped and evolved.

The proposal originates in required support for the initial phase of the Agile Modeling Method Engineering (AMME) methodology, which was successfully applied in developing a variety of project-specific modeling tools. From this accumulated experience, awareness of "modeling method requirements" emerged and informed the design decisions of CoChaCo.

***Index Terms* — Modeling method requirements, Requirements modeling, Metamodeling, Agile Modeling Method Engineering**

I. INTRODUCTION

Modeling methods are often taken for granted (as de facto or de jure standards). However, they are artifacts subject to specific engineering methodologies and driven by a particular class of requirements - to be labelled in this work as *modeling method requirements*. The representation and management of this class of requirements, as well as their relation to subsequent method engineering phases, reclaim tool support and methods of adequate specificity, considering their characteristics and building blocks, their specific engineering cycles.

The existence of such requirements has been occasionally implied by practices such as situational method engineering [1], domain-specific language engineering [2][3] or Agile Modeling Method Engineering (AMME) [4] – the latter being the encompassing framework whose observed application motivates the proposal of this paper. The notion of "agility" is tightly coupled with that of "requirements" – i.e., enabling agility in modeling methods means enabling responsiveness to requirements that may be situational, domain-specific, enterprise-specific, evolving, etc. Addressed agility cases include: the implementation of an existing modeling standard, the enrichment or hybridization of standards, the development of new modeling methods for which a standard is not available, or the evolution of an already in place modeling method (triggered by evolving requirements).

For all these cases, AMME provides a conceptualization framework that revolves around the underlying notion of "modeling method" [5] comprising the building blocks: (i) *Modeling language* - itself decomposed structurally into notation, syntax, semantics, possibly partitioned into multiple "viewpoints" if the language grows too complex for a single type of diagram; (ii) *Mechanisms* (including algorithms) - comprising all functionality that operates on model contents, to satisfy relevant modeling use cases (e.g., code generation, report generation, model transformation etc.); (iii) *Modeling procedure* - i.e., how the method should be used, considering its purposeful nature (all modeling intentions, capabilities and use cases).

Elicitation, representation and management of requirements for each of these building blocks are challenging but weakly supported. The AMME framework includes a conceptualization process (to be detailed in Section III) that starts with the *Create* phase, covering all the pre-design efforts. Compared to the subsequent AMME phases, which benefit from good support in terms of fast prototyping platforms and deployment enablers, this *Create* phase (and partly its subsequent *Design* phase) did not have, until this proposal, a clearly articulated support. The authors' longitudinal observation of repeated applications of AMME (between different projects or different iterations of the same project), led to the motivation of devising a specific solution to support the *Create* phase.

Furthermore, by applying AMME onto itself, the proposed solution came to be a modeling method (and corresponding tool), an idea that leverages the benefits that conceptual modeling brings to requirements management – e. g., streamlining semantics towards subsequent development phases [6], user-friendly knowledge capture [7]. The result is labelled with the acronym CoChaCo (Concept-Characteristic-Connector).

The remainder of the paper is structured as follows: Section II will establish the working terminology for this paper and clarify the role of various enablers. Section III will formulate the problem statement and will provide a summary of the proposed solution. Section IV will dissect the notion of "modeling method requirements" - the underlying motivation of this work. Section V will provide details on design and implementation details of the proposed modeling method. Section VI will illustrate the viability of the solution in project-based cases. Section VII will comment on related and predecessor works. The paper ends with concluding evaluation insights and an outlook to future development plans.

II. Working Terminology and Enablers

Modeling method requirements are the specific class of requirements that motivate the engineering of conceptual modeling methods. A taxonomy of such requirements is necessary to enrich the RE body of knowledge and to inspire the development of dedicated support – the work at hand being an initial step in this respect.

CoChaCo is the key artefact proposed by this paper – a modeling method whose current implementation is labelled *CoChaCo4ADOxx* (hinting to the underlying development platform). Its goal is to facilitate the documentation and analysis of modeling method requirements and to streamline the supported RE effort with subsequent phases of modeling method engineering.

ADOxx [8] is an open access metamodeling platform on which a diversity of domain-specific modeling tools have been implemented [9]. This is both the platform *on which* the current prototype of CoChaCo was implemented and, at the same time, the platform *for which* it was prototyped. However, the coupling between CoChaCo and ADOxx is rather flexible: (i) core concepts of its modeling language can serve domain modeling in the most generic sense, (ii) additional concepts are specific to modeling method requirements management (e.g., Purpose, Functionality, Stakeholder, relationships between them), independently of how such a method will be implemented); (iii) certain aspects are ADOxx-specific to support some development streamlining (e.g., ADOxx attribute types rather than the MOF standard [10] and other mechanisms - see Section V).

AMME is an agile methodology for developing modeling methods. From its past applications, several meta-requirements emerged and motivated this work – i.e., CoChaCo was developed to fill certain gaps in streamlining AMME phases, as detailed in the next Section.

III. Problem Statement and Solution Summary

AMME has been successfully employed in a number of projects where agile modeling tools had to be developed for a variety of goals - some *educational* (e.g., hybridizing multiple fundamental modeling languages in the same tool [11]), some *research-oriented* (e.g., to enable domain-specific knowledge conversion [12]). Here the term "agile" applies not only to model contents and the activity of modeling, but also to the methods and tools that must (co-)evolve according to changing requirements, or must be tailored for project-specific purposes (e.g., to capture richer knowledge than what a standard or modeling technique allows, to ensure interoperability with model-driven systems). The operationalization enabler for AMME are the fast prototyping platforms (see [8][13]) - however the development tasks are integral part of an engineering cycle (depicted in Fig. 1) that requires adequate streamlining, along the following phases:

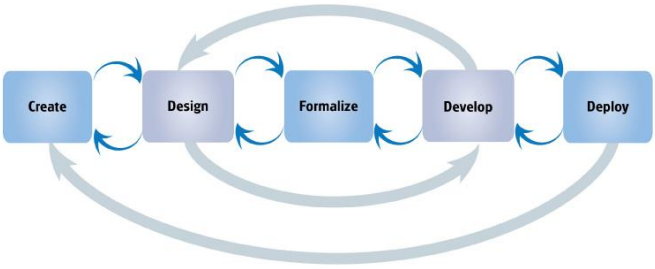

Fig. 1. The AMME conceptualization lifecycle [4]

Create: the opportunity of a modeling method is identified, including modeling scenarios, goals and stakeholders. Modeling method requirements and domain knowledge guide the reduction of the application domain to a "knowledge schema" that will go through a gradual refinement process - from the early stage maturity of a "domain model" to the operational maturity of a "modeling language vocabulary" (metamodel).

Design: the modeling method building blocks (language, mechanisms, procedure) are designed and specified on an adequate level of detail depending on application and reusability goals (e.g., platform-specific or platform-independent). The early stage metamodel becomes a language vocabulary enriched with grammar, machine-readable semantics and semiotics (interpreted notation).

Formalize: formal lenses are applied to the method specification in order to stimulate scientific scrutiny, knowledge questions and dissemination. This phase may be skipped due to pragmatic project constraints; even in these cases, a certain degree of rigor is already imposed by the underlying formalism of the metamodeling platform of choice – e.g., for ADOxx, this phase can fall back on the platform-specific FDMM formalism [14] (it defines the notions of model, model type according to the meta-metamodel underlying all ADOxx implementations).

Develop: a usable modeling tool is developed on a metamodeling platform allowing knowledge engineers to focus on the modeling method building blocks, while reusing built-in generic functionality (e.g., model storage, look-and-feel of the drawing canvas).

Deploy: the modeling tool is deployed (on desktops, in the cloud, with remote access, etc.) and evaluated in use. This phase feeds back into *Create* due to evolving requirements – the evolution may be (i) extrinsically motivated (new analysis

scenarios needed, new model-driven systems must be built) or (ii) intrinsically motivated (non-experienced users gradually bring new ideas on how the custom modeling tool can support them, thus triggering short update cycles).

The research challenge addressed by this paper is *How to support the Create phase of this lifecycle in a way that enables streamlining with other phases?* In many AMME projects this phase was more of a knowledge acquisition effort. It became obvious that an RE perspective is necessary considering the need for an articulated integration of AMME phases.

In deciding the nature of the RE method to be developed, we have been inspired by the design research paradigm. The work started by distilling several high-level meta-requirements observed in past AMME projects, summarized in Table I. Each meta-requirement is mapped on qualities of agile modeling methods, including references to publications and projects where those aspects have been emphasized.

In response to these meta-requirements, we posit that agile domain-specific modeling may be employed to satisfy these meta-requirements - this had led to the decision that CoChaCo itself should be a modeling method, engineered according to AMME and having a first prototype developed in ADOxx (in other words, we applied AMME onto itself).

The proposed solution was therefore designed as a modeling method that integrates ideas from (i) early stage requirements modeling (e.g., goal modeling, use case modeling) with (ii) process modeling (referring to the target method's application procedure) and (iii) metamodeling (an early stage knowledge acquisition effort that blends into the *Design* phase of AMME).

TABLE I. META-REQUIREMENTS FOR REPRESENTING MODELING METHOD REQUIREMENTS

The meta-requirements	How agile modeling methods can fulfil the meta-requirements
Modeling method requirements should be represented in ways that facilitate communication, annotation and understanding (both zooming in and high-level overview).	Conceptual modeling traditionally aims to enhance communication and understanding, whereas domain-specific modeling implies a semantic enrichment of model elements. For complexity management and comprehension AMME recommends a navigable decomposition of a modeling language, successfully applied in multi-view enterprise modeling [15]. Generic consistency management frameworks are being investigated to support such decompositions [16].
Modeling method requirements should be sufficiently granular to inform the *Design* phase, producing an early stage metamodel that will be later refined in a modeling language vocabulary.	AMME allows a language engineer to customize semantic granularity both on concept level (multiple levels of specialization or multi-level modeling) and inside a concept (custom property sheets for each modeling element). The proposed CoChaCo method aims to produce a machine-readable metamodel that can be adopted as a starting point for refinement in the *Design* phase.
Modeling method requirements should be distinguishable and traceable by specific taxonomies, not only by generic RE taxonomies (functional, nonfunctional etc.)	The modeling method notion [5] specifies building blocks on which requirements may be mapped. Existing experience with AMME suggests the typical change propagation paths among those building blocks.
Modeling method requirements should be valued and manipulated as "requirements knowledge" assets [17] – i.e., a knowledge management approach must complement the RE effort.	Agile modeling can be treated as a knowledge conversion process (see [12]) - a possible specialization of Nonaka's SECI model [18]. We also refer here to the interweaving of RE and architectural concerns recognized by the literature [19] which is translated here in the dependencies between AMME's *Create* phase and its adjacent phases.
Modeling method requirements management should be supported by flexible systems where the annotation / documentation schema can be easily extended, reused and can interoperate with other systems that may need to consume the requirements knowledge.	In the context of AMME, agility has been defined as an amalgamation of the following qualities: (a) *adaptability & extensibility* (the possibility to change or extend existing modeling methods or fragments), (b) *integrability* (the possibility to integrate multiple modeling languages or fragments, or the modeling environment with external systems that must read model contents), (c) *operability & usability* (the possibility to include functionality that enhances the modeling experience and model comprehension). An example of a tool that demonstrates these qualities is BEE-UP [19], integrating known languages such as BPMN, UML, ER, Petri Nets and EPC.

IV. TOWARDS A TAXONOMY FOR MODELING METHOD REQUIREMENTS

Beside the provided operational solution, a secondary goal of this paper is to highlight the notion of *modeling method requirements*. While the traditional taxonomical categories (functional, non-functional, etc.) [21][22] are also applicable for modeling software development, this class of requirements must be acknowledged, both for representation and traceability purposes, in projects where modeling products are developed.

A rich collection of domain-specific modeling tools and methods have been catalogued in the literature [9]. The Open Models Laboratory [23][24] provides a community hub and digital ecosystem for the conceptualization and dissemination of such artifacts. The authors' project experience helped synthesizing some distinguishing characteristics of modeling method requirements.

Firstly, a taxonomy can be derived by the modeling method building blocks specified by Karagiannis and Kühn [5], characterized and exemplified as follows:

Language requirements cover all requests regarding model contents – i.e., notation, syntax and semantics. Examples: "I want to use this particular icon, specific to my company culture" (*notation requirement*), "I want to capture this aspect in a separate diagram type to avoid visual cluttering" (language partitioning, included under *syntax requirements*), "I want to be able to attach risk levels/severities to my BPMN tasks", "I want the actors in my organization to be distinguishable by gender" (*semantic requirements*).

Mechanisms requirements cover requests on the functionality available in the modeling tool. Examples: "I need to be able to generate this kind of report from my models ", "I need to be warned by graphical highlighting when this semantic condition is fulfilled".

Procedure requirements cover requests on how models can be created, including usability requirements and general model-

ing experience. Examples: "This diagram element should be inserted automatically", "This type of diagram should be generated from my data logs", "This model should be created and annotated collaboratively, by people in two departments".

It is, of course, debatable to what extent these examples qualify as requirements (i.e., explicit needs) or as forespoken solutions (i.e., design specifications in support of implicit needs). This status may shift from one AMME iteration to another – as users get accustomed with hands-on experience they would raise explicit change requests that blur the distinction, e.g., going from "I want a more expressive domain-specific notation" to "I want this particular icon that corresponds to this aspect of my enterprise's culture". Traditional classes of requirements are also applicable (e.g., a functional requirement is typically a requirement for a mechanism) – however, an explicit mapping on the specific building blocks reduces ambiguity, improves traceability and creates opportunities of streamlining.

Moreover, this taxonomy helps revealing some *change propagation paths* as identified and summarized in Fig. 2, which are necessary to establish dependencies between backlog items during the development phases.

The figure considers the generic characteristics of a modeling method – not only its building blocks (language, mechanisms, procedure) but also its typical usage (either as a modeling tool, or as a modeling environment attached to some model-driven systems). The general case of a multi-view method is considered in the figure – i.e., a language comprising multiple types of models to reduce visual cluttering and to separate concerns, while preserving cross-view relations to support consistency management [25] (in the area of enterprise modeling, the complexity of the systems under study is associated with the need for multi-perspective modeling [26]).

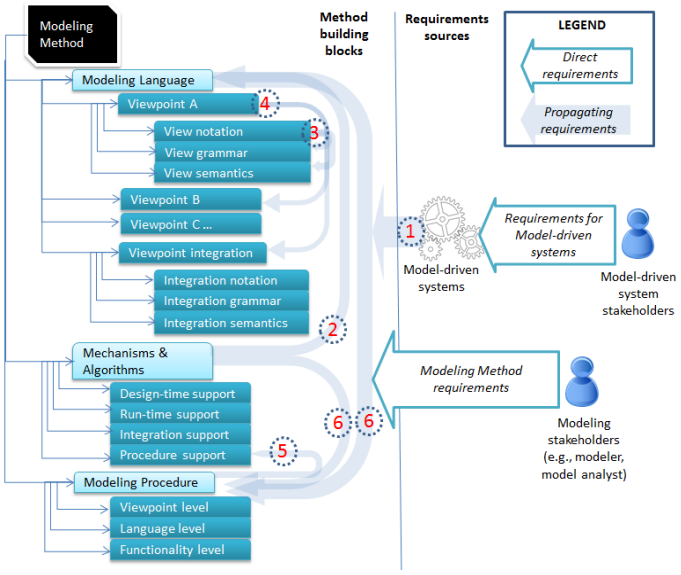

Fig. 2. Modeling method requirements propagation paths – adapted from [15]

Two sources of modeling method requirements are identified: (i) those stated by users that can relate directly to the modeling experience and tool usage; and (ii) those derived indirectly from requirements raised for some model-driven systems that interact with model contents (e.g., via code generation, model queries or other kind of interoperability bridge).

AMME reverses the dependency between models and model-driven software, enabling software engineers to raise requirements for tailoring the modeling method, thus enriching the software's database/knowledgebase tier and its semantic space (see [27] for a proposed software engineering method based on this principle).

The typical change propagation paths, as numbered in the figure, are:

1. *Requirements for model-driven systems* can propagate in semantic requirements, reclaiming an extension of the modeling language semantic space (e.g., new properties, new concepts to be made available for model queries).

2. The same applies to *requirements on mechanisms* - since they take input from model contents, a sufficiently rich semantic space must also be ensured for the required mechanisms.

3. *Requirements on notation* can be volatile, as users perceive models primarily on a visual level and will want to benefit, once they understand the opportunity provided by AMME, from customizing notation according to some local enterprise culture, personal preferences for comprehension or interactivity of the graphic layer. The business process modeling literature has recognized the notion of "secondary notation" and the nudging effect it can have on comprehension [28]. Dynamic or interactive notations are rule-based, therefore driven by semantics (i.e., the presence of some property or instance in the current model). Visual cluttering may also be addressed by customizing notation.

4. Changes in *requirements pertaining to a viewpoint* (model type) can propagate in other viewpoints to ensure consistency or complementarity, depending on the existing cross-view relationships (manifesting as hyperlinks or model synchronization rules). For example, to reduce linking effort or to shift the "border" between viewpoints, concepts may be moved between different types of models, or artificially introduced as workarounds.

5. The *modeling procedure* guides the user in creating purposeful models. This building block is not always acknowledged explicitly, but certain requirements point to it – e.g., "I don't want to have more than x types of models", "I don't want to create this kind of model, it should be generated automatically as a map of the existing models". Such requirements will typically propagate towards a need for additional mechanisms (then further towards semantics).

6. Finally, any changes in modeling language or functionality will affect the modeling procedure (and any guiding documentation that is built for it).

Recognizing a modeling method requirements taxonomy and related propagation paths is a first step towards a modeling-oriented practice of RE, for which the hereby proposed CoChaCo method can be an operationalization enabler.

The next sections will focus on the design decisions of this method and illustrative examples.

V. Design Decisions and Implementation Details

The CoChaCo4ADOxx tool was engineered by following the AMME methodology and implements the proposed method's building blocks, as described in this section.

A. The Modeling Procedure

The modeling procedure is depicted on the left side of Fig. 3, together with some sample models for which key CoChaCo concepts are visible. These are for a toy example of a "cooking modeling method" - designed to support knowledge management for a festive dinner planner. We start with this kind of example in order to detach the proposal from a Software Engineering context, and to emphasize the general value of modeling as *means of knowledge representation* (for which code generation, software documentation or business process analysis can be considered domain-specific use cases). This perception on model value, discussed in more detail in a value co-creation context by Strecker et al. [29], is what makes modeling also adequate for requirements representation.

The modeling procedure leads the modeler through a structuring and abstraction effort that is partly inspired by traditional knowledge acquisition techniques, adapted to the specificity and building blocks of modeling methods. The motivating assumption is that stakeholders acknowledge the need for a modeling tool. We expect that CoChaCo would also be useful in conceptualization efforts aiming for a different outcome (i.e., domain analysis) but this paper's scope is limited to the primary goal of supporting modeling method/tool engineering.

The procedure steps that belong to AMME's *Create* phase are as follows:

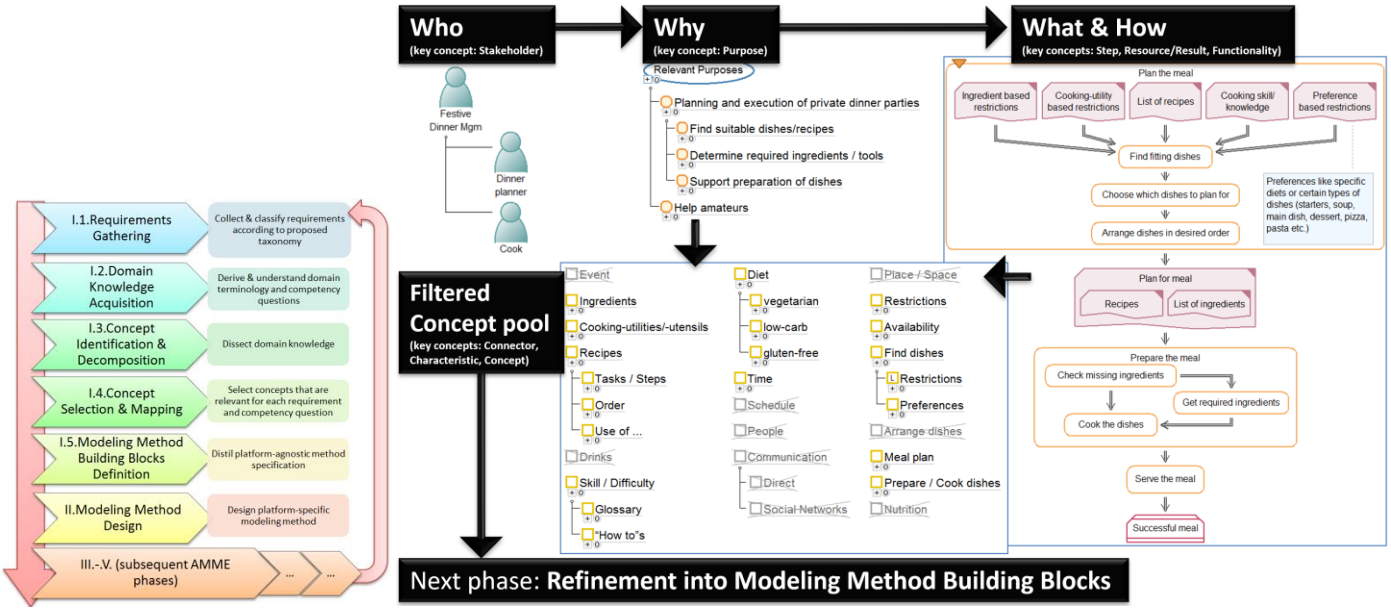

Fig. 3. The modeling procedure (left) and model samples supporting the first four steps (right)

Step 1. Requirements Gathering may involve traditional RE approaches enhanced by the CoChaCo modeling tool for loose note taking and weakly structured mind mapping. This step primarily aims for interaction with stakeholders and documenting their wishes in order to identify modeling goals, rationale and scenarios. Both *direct stakeholders* (that will operate on models) and *indirect stakeholders* (that will use model-driven systems) should be considered and can be represented – i.e., associated through machine-readable relations (cross-diagram hyperlinks) with the CoChaCo elements they "require".

Clarifications about how models can help stakeholders are needed at this stage of the modeling procedure, since it is often the case that models are assumed to be graphical documentation (e.g., an alternative to Powerpoint diagrams). CoChaCo shifts this perception towards understanding the value of models as knowledge representation – if this is not clear for all stakeholders in the initial iteration, the next iterations of AMME make it evident through hands-on experience with throwaway prototypes. Examples of features that help with this perspective shift are *model queries* (knowledge is there to be queried, not only to fancy up some reports), *rule-based behavior* (e.g., dynamic notation based on semantic changes) or *interoperability features* (e.g., generating something from models, coupling the modeling environment with some external system). Throwaway prototypes should showcase one or the other of such features as early as possible to stimulate the refinement of modeling method requirements.

Step 2. Domain Knowledge Acquisition aims to derive domain understanding and expertise. This step may involve the traditional approaches reviewed by the literature [30][31] (e.g., laddering, card sorting, "20-questions"), however it will gradually focus on the competency questions that models should be able to answer, derived from stakeholder goals and their now enhanced understanding of "model value" - for humans, for systems (e.g., process-aware systems [32]) or for both. Termi-

nology is clarified at this stage and collected around competency questions or "Five Ws + How" questions.

Step 3. Concept Identification and Decomposition. Decompositions of goal statements and competency questions will lead to more refined CoChaCo models. It should be clarified at this step what knowledge will be externalized through models and what criteria will inform decision-making. This step will link stakeholder purposes to work flows, informational resources and finally collections of key terms.

Step 4. Concept Selection and Mapping. The key terms are filtered to keep the vocabulary concise and limited to the explicit purposes derived in earlier steps. The depth of domain-specificity is decided, while at the same time keeping options open for future agile iterations (irrelevant terms are scrapped rather than removed, hierarchies are kept open and populated with broader superconcepts, relationships are weakly constrained).

Step 5. Modeling Method Building Blocks Definition. Before targeting a specific platform, the modeling method building blocks are mapped at this stage on the corresponding requirements taxonomy - key terms from the earlier step are mapped on language constructs, on procedure steps or on mechanism requirements.

This last step blends into the *Design* phase by providing an early stage modeling method structure, including a metamodel built from the concepts selected in the preceding step. This method specification will be platform-agnostic, needing further specialization for platform-specific constraints and features. Examples of platform-specific design decisions are (for ADOxx): which platform-specific datatypes should be applied, which metamodel partitions will become actual model types and which will be object repositories; which relations will become connectors and which will end up as hyperlinks, which mechanisms should run inside the modeling environment and which will be external plug-ins.

B. The Modeling Language

The CoChaCo language comprises a minimal set of meta-constructs for building an early stage, platform-agnostic metamodel:

- the *Concept*, often ending up as a graphical symbol of the language (although it may also become a non-graphic object involved only on a functional level, or a tabular attribute);
- the *Characteristic*, often ending up as an editable attribute (although it may also become a graphic characteristic or connector);
- the *Connector*, often ending up as a visual connector (although other options are also available - e.g., hyperlink, containment relation).

The general principle of CoChaCo is that these core constructs do not have a prescribed mapping on a platform-specific implementation or style of modeling. They are rather ontological constructs whose manifestation in the implemented modeling tool will be decided later – therefore this step is also included here in the *Create* phase, while having a close coupling with AMME's *Design* phase (which is forward looking at the implementation prospects).

For the other method building blocks, additional meta-constructs are available and linkable to the core constructs:

- the *Purpose* (goal);
- the *Functionality* (solution that satisfies the goal and is dependent on particular language constructs);
- the *Step* (of a procedure or mechanism);
- the *Resource/Result* (needed or produced by a Step);
- the *Stakeholder* (a modeling actor or model-driven system that motivates an aspect captured in CoChaCo – a concept, a purpose, etc.).

Visual meta-connectors are also looser than in typical domain modeling – e.g., an *asemantic relation* is also allowed for loose diagramming in the style of mind mapping (semantics to be decided later or improvised through annotation); a *flow/order relation* covers both control flow and resource flows. The major constructs are visible in the legend of Fig. 4, where CoChaCo was applied onto itself to give a high-level view on its own metamodel. The loose semantics are intended to establish a balance between imposition and flexibility, to achieve a degree of open-endedness that allows agile reconsideration of the nature of language constructs and the versioning (with minimal editing) across multiple iterations.

For this reason, the following visual meta-connectors restrictions are recommended (through warnings, model queries or domain/range checks) rather than enforced by the metamodel in Fig. 4:

- *connects* is recommended to be used between a Connector and the Concepts it should connect (may also be an n-ary connection whose implementation-level nature will be decided later);
- *flow* is recommended to indicate the control flow between procedural/functionality steps or the flow of resources into/out of a step (example visible in Fig. 3, not captured in the metamodel's legend);
- *hierarchy* and *specialization* are distinguished to allow for an asemantic hierarchical note-taking (mind mapping style) before starting to think if the hierarchy is an actual subsumption or just an intuitive attempt of the stakeholder to hierarchically structure his/her thoughts; the strict specialization should be used between constructs of the same kind;
- *uses* is recommended to link Purpose, Functionality or Step to something that it relies on (could be another functionality, but also a concept or a characteristics it depends on);
- *has* is recommended to suggest ownership or partonomy (e.g., a concept has a notation, or contains another concept whose instances cannot exist by themselves);
- additionally, a *relevant_for* hyperlink relates any CoChaCo element to the stakeholder who needs it (in Fig. 4 the entire model set is attached to a modeling method engineer).

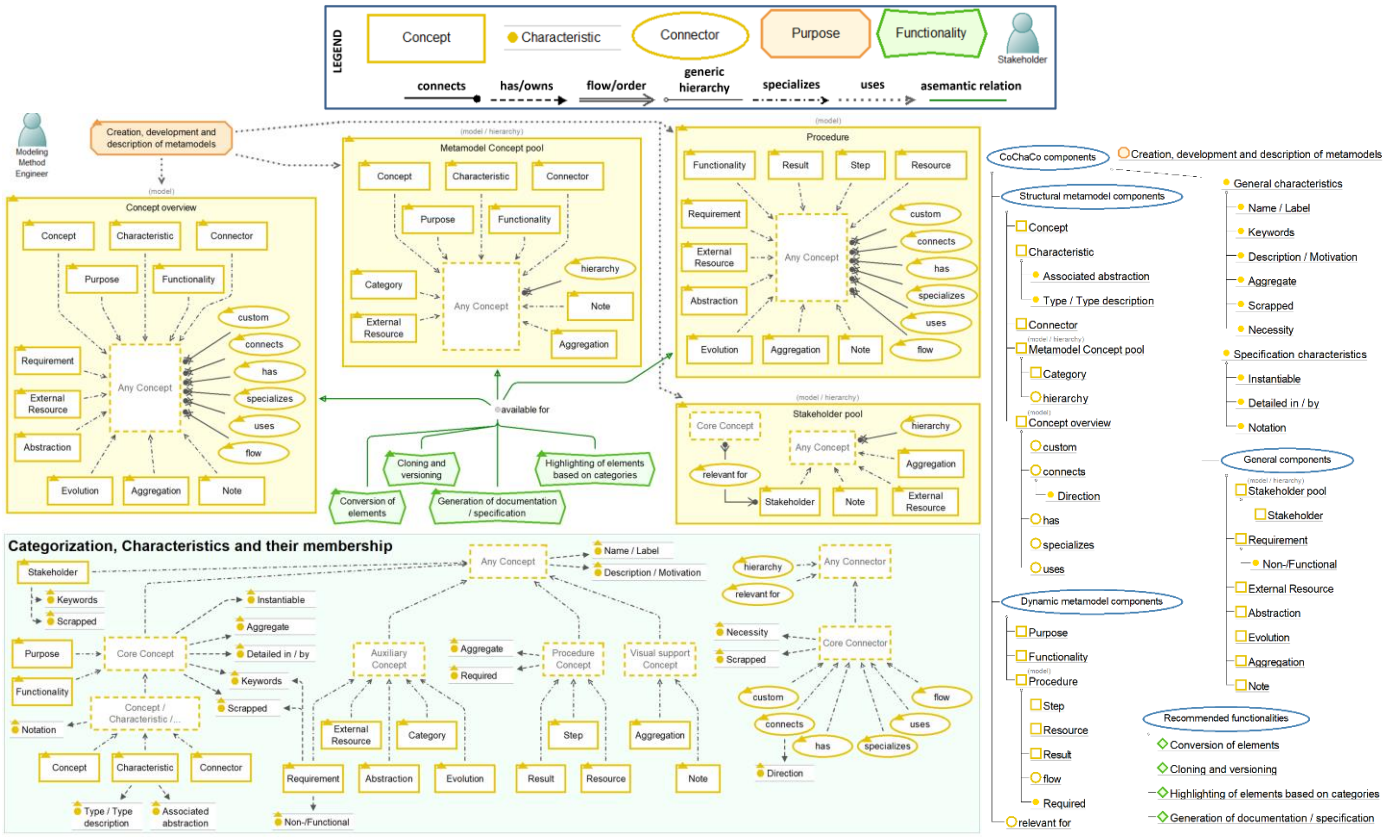

Fig. 4. The CoChaCo metamodel self-described in CoChaCo terms

All these interpretations reflect recurring patterns found in earlier AMME projects during knowledge acquisition efforts based on loose notetaking (e.g., mind mapping) rather than strict UML domain modeling. Thus, these patterns aim to establish a sensible middle ground between flexibility and formality, one that emerges from how modeling tools are perceived by stakeholders rather than how they could be described in UML terms – i.e., the *has* meta-connector does not necessarily end up becoming a UML aggregation or composition; instead, it relates to the user perception that, for example, a concept "should have" a graphical symbol, or that a connector "should have" a direction. It is expected that future iterations of CoChaCo will enrich this list of thinking patterns, as the project portfolio where the method is applied will expand.

Other constructs visible in the metamodel can be loosely attached for annotation, versioning and notetaking purposes in the early steps of the modeling procedure (e.g., *Evolution*, *Requirement*, *Note*, dependency on *External Resource*).

As an illustrative example, we show in Fig. 5 how the language requirements for a Petri Nets modeling method evolve through three different versions by gradually adding to them syntactic requirements, semantic requirements and mechanism requirements. Moreover, the preferred graphical symbols (notation requirements) are also linked to the CoChaCo "Concepts". As mentioned before, the distinction between requirement and solution will sometimes be blurred and will shift depending on stakeholder familiarity with modeling. Typical cases are: (i) *requirements by analogy* ("I want to have a language like BPMN plus a concept of Risk according to my business-specific taxonomy"); (ii) *requirements that enforce solution* ("I want to be able to annotate this element with this list of attributes that are relevant for my knowledge management position"); (iii) *vague requirements* ("I want to be able to design my cooking recipes, show me a throwaway prototype and we'll discuss what should be added on that").

This variability in the nature of modeling method requirements is one key aspect that CoChaCo aims to agilely support.

C. Mechanisms

The current scope of the reported solution is to support the management of requirements and domain knowledge in AMME's *Create* phase. In the long term, certain mechanisms are planned to streamline AMME phases by generating out of CoChaCo models a machine-readable modeling method specification (currently a quite heavy document that must be redacted by method engineers). This should be platform-specific input for rapid prototyping.

In the current implementation, such support is limited to the following features:

A. *Compatibility-checking scripts* that report the deviations existing between the designed metamodel and platform-specific constraints. Currently, the ADOxx platform is supported (examples of constraints that are verified: if ADOxx datatypes have been used, if each connector has exactly one domain and one range).

B. The possibility to *export CoChaCo models as RDF knowledge graphs*, thus exposing them to potential model compilers that can produce various model-driven artifacts (e.g., traceability reports). This is based on a model RDFiser plug-in available for tools developed on the ADOxx platform (e.g., it is currently integrated in the BEE-UP tool to enable RDFisation of BPMN, UML, EPC, Petri Nets and ER models [8]).

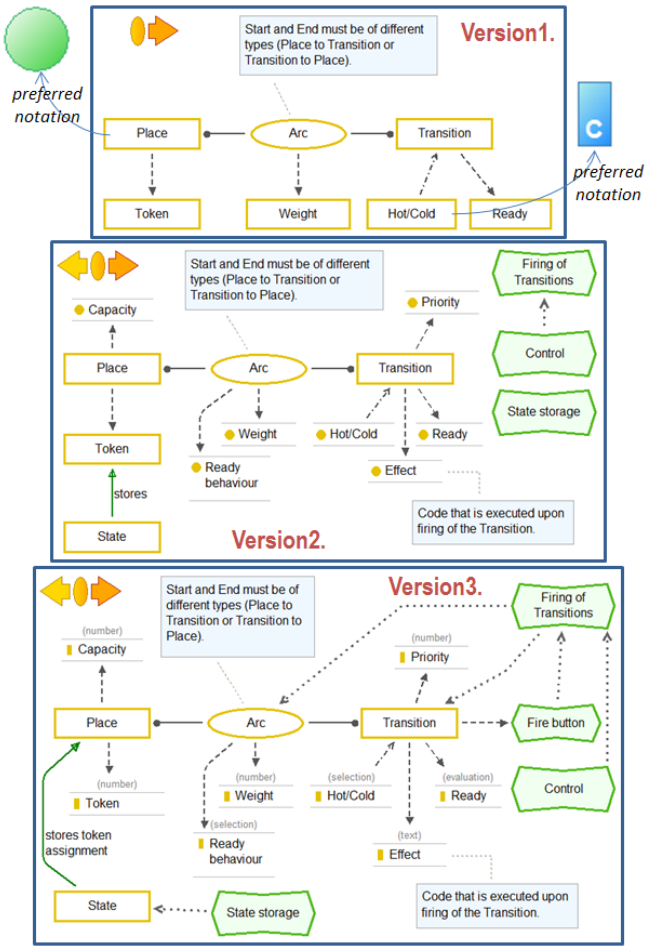

Fig. 5. Evolving language requirements into an early stage metamodel for Petri Nets

In addition, a mechanism for generating specification documents is included. It automatically populates Open Document templates with model contents and annotations that a developer would need to start prototyping the modeling tool. Other mechanisms are the usability or model management features – highlighting, versioning, cloning.

Furthermore, some generic mechanisms are provided by the underlying metamodeling platform, agilely repurposed for the CoChaCo language – model queries, model exports and model comparison, all support RE scenarios pertaining to documentation, traceability or report generation. For example, model queries can track propagations of changes – either inside the modeling environment (using the underlying platform's AQL graph query language) or outside the modeling environment, on the RDF knowledge graphs exported from models.

VI. Selected Application Case

An application case will be illustrated in this section, based on a European project for which a domain-specific modeling method was developed. One purpose of that modeling method was to support the *process-centric documentation of mobile app requirements*; this later evolved to an aspiration *to generate app orchestration flows out of process models* – those flows would then interoperate with an orchestration engine to actually deploy chained mobile apps according to the modelled process flow (assuming a "bring-your-own-device" industrial setting).

Fig. 6 shows an early iteration of the process modeling language – one where mobile app requirements were rudimentary text annotations attached to business process tasks, in order to support reporting functionality.

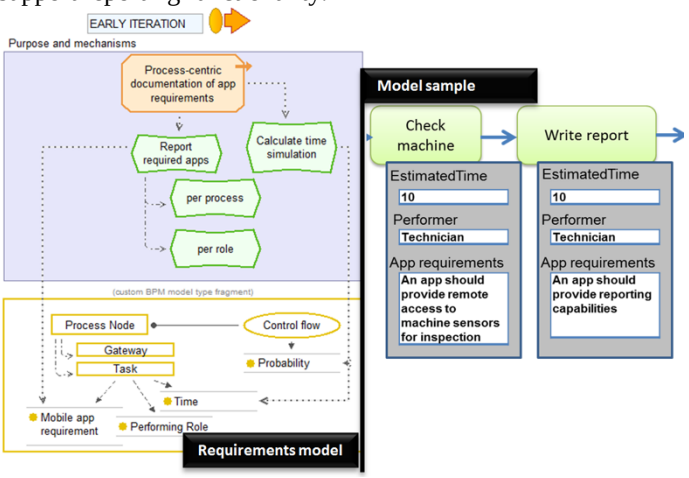

Fig. 6. Early iteration method requirements (left) and model sample (right)

The interpretation of this example is: a maintenance app engineer (*stakeholder*, not visible as it is hyperlinked from outside the model) has the *purpose* of documenting app requirements in model form; for this, he/she needs the *functionality* of reporting required apps per process or per role, which relies on the ability to attach to the Task *concept* a Mobile app requirement *characteristic*. In addition, a *functionality* for calculating time simulation needs Probabilities as a *characteristic* of Control flow *connectors*, and a Time attribute as a *characteristic* of Tasks.

Fig. 7 shows a late iteration where the app requirements modeling technique was significantly expanded. One can notice the following features of CoChaCo:

- The scrapped (but still preserved) language constructs, since the app requirements representation evolved from the rudimentary text annotation to a distinct type of model where apps could be described as mockups and attached via hyperlinks to the process tasks where they were required;
- The partitioning of the concepts in implementation recommendations (different types of models connected by hyperlinks), including the newly required type of model that had to be generated (mobile app chains/orchestrations);
- The mechanism requirements (generate app orchestration then export model) mapped on (i) the purpose of

interoperating with the model-driven orchestration engine; (ii) the language constructs on which this functionality depends; (iii) and a possibility of linking it to design or implementation artifacts to be developed in later AMME phases.

The underlying graph query engine will facilitate queries on a model or across multiple linked models, which can be tailored in order to obtain reports that are relevant for the RE scope. Examples:

a. Give me all characteristics that are used by all functionalities involved in achieving the purpose "Interoperate with orchestration engine". The query expression in the platform's native AQL syntax is:

```
((({"Interoperate with orchestration engine":"Purpose"}->"uses")<-"flow")->"uses")>"Characteristic"<
```

b. Give me all functionalities to which changes in the Process Node can be propagated – either directly (via the uses relation), or indirectly (via the connectors).

```
(({"Process Node"}<<-"uses")>"Functionality"<)OR
(({"Process Node"}<-"connects)<<-"uses")>"Functionality"<)
```

c. Give me all functionalities that rely on the characteristics that have been scrapped

```
(((<"Characteristic">[?"Scrapped" = "yes"])<-"has")<-"uses")
```

Templates based on such queries are realized for reporting or traceability purposes. Such templates are prepared on a project-specific basis as they imply a learning curve. The query language is, however, part of the basic skillset for AMME engineers working with the ADOxx platform and is a conveniently flexible mechanism for requirements analysis, or the further development of a toolset that uses CoChaCo models as requirements knowledge.

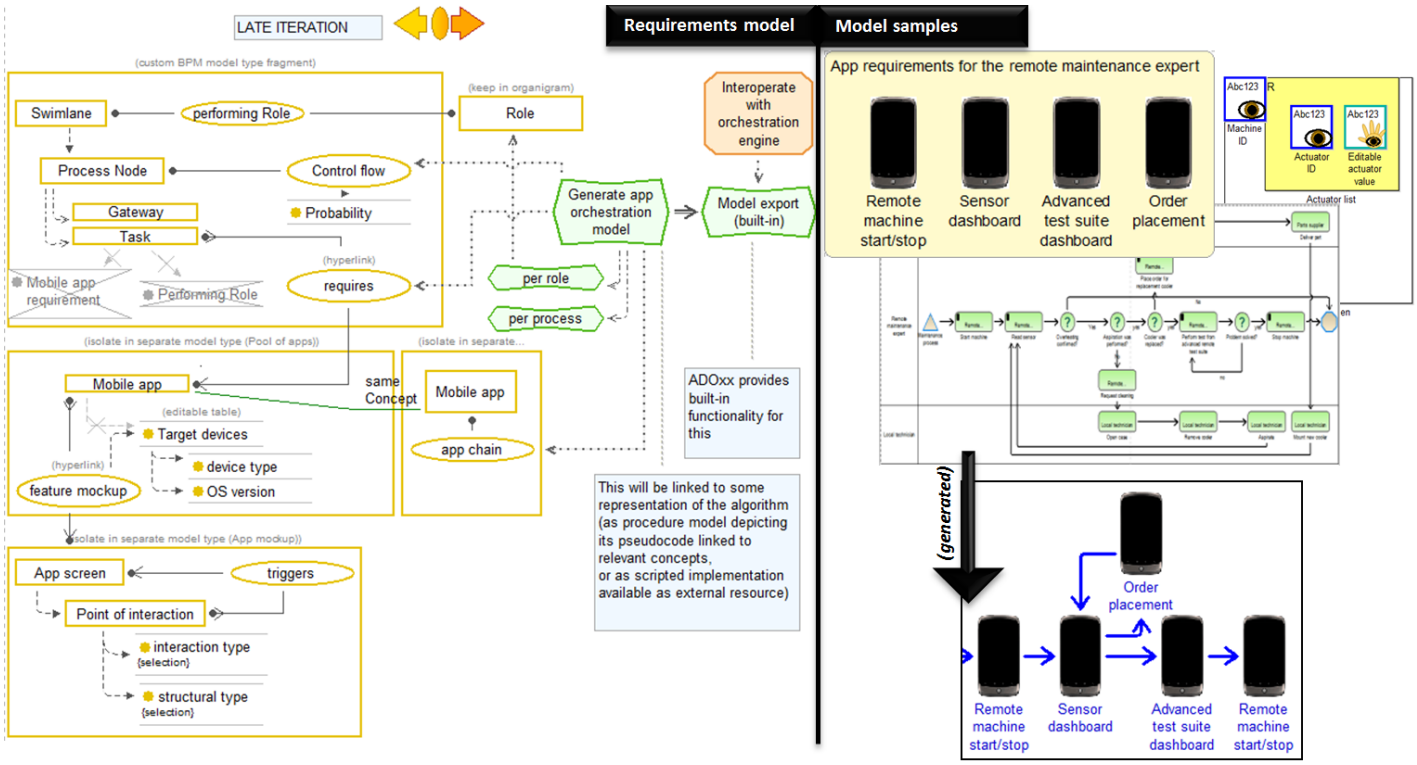

Fig. 7. Evolved method requirements (left) and sample models (right)

VII. Related Works

RE research roadmaps are periodically proposed and revised (see [33]) however the class of modeling method requirements is rather neglected or only indirectly suggested in the context of method engineering - e.g., Ralyte mentions "engineering intentions" that a situational method should support [34]; Henderson-Sellers et al. proposed intention achievement guidelines based on method knowledge (graphs of intentions and strategies) [35]; the work of Gupta and Prakash relates closer to our work by defining *method requirements* as "high-level abstraction of services that a method will provide and constraints under which it functions" [36]. This definition may supersede our notion of modeling method requirements, which adds specificity derived from the definition of a modeling method as employed in this work. Requirements on domain-specific languages have been characterized [2] as *generic* or *specific*, and meta-requirements for documenting and analyzing them are raised there – our proposal is a possible solution in this respect. The work of Moody on modeling notations [37] implies the idea of notation requirements by proposing a design space based on several variables and qualities.

Regardless of these predecessor notions and their semantic overlaps, a gap needs to be filled in terms of operationalization support and dedicated RE methods for this class of requirements. In this respect, our work makes a proposal derived from accumulated practice with applying AMME in the development of domain-specific modeling tools.

The design decisions of CoChaCo aim for a balance between formality and flexibility, being inspired by the interplay between conceptual modeling (i.e., modeling governed by formal concepts) and loosely-interpreted mind mapping (i.e., graphical note-taking). This is in turn inspired by a proposal of interplay between information retrieval and mind mapping [38], recently instantiated in works that aim to diminish the gap between domain analysts and software designers by streamlining conceptual modeling and mind mapping [39][40]. Such works are addressing general-purpose model-driven engineering, while our proposal is distinguished by (i) being narrowed to the specificity of modeling tool development; (ii) employing a customized modeling language intended for modeling method requirements management, instead of repurposing UML class diagrams (by doing this we aim for the benefits of domain-specific languages [3]).

Requirements models have been subjected to ontological evaluation (via reasoning) or ontology-driven specification [41][42]. By employing a metamodeling platform to implement a requirements modeling method we enable comparable benefits enabled by the platform of choice – i.e., the accumulation of a diagrammatic knowledge base with constraints imposed on scripting level or by a metamodel compliant with the platform-specific formalism (see [14]).

VIII. Concluding Evaluation and Future Work

The paper introduces the CoChaCo method and its implementation for modeling method requirements engineering. It also emphasizes the relevance of this specific category of requirements (rarely recognized by RE) that drives modeling method engineering processes. The proposed method was evaluated for its viability, by deploying it in the form of a toolset comprising (i) a modeling tool (CoChaCo4ADOxx); (ii) interoperability mechanisms targeting the ADOxx development platform and the generation of specification documents.

The evaluation strategy included both checking the retrofit of the proposal to past projects (when CoChaCo was needed but not available) as well as applying it to on-going projects. The evaluation was guided by the reference criteria for requirements methods [43]:

How does the method fit into the development process? CoChaCo was from the very beginning engineered to fill a gap in an existing development model (AMME), therefore it has strong familiarity with the way of thinking of AMME practitioners. Its implementation benefits from repurposed features of the underlying ADOxx platform – e.g., graph-based model queries and scripts allow the discovery of significant relations and items in the produced artifacts, the generation of documentation and consistency checks. One key meta-requirement that is not yet satisfied in this respect is the ability to generate a platform-specific definition of a modeling method.

Ease of use. Ease of navigation across semantically related diagrams is facilitated by the underlying ADOxx platform through its default look and feel, a modeling assistant and model browser. From current observations, for AMME practitioners, CoChaCo reduces by half the time spent on writing and managing a method specification document – this is an estimation based on a single on-going project (a long-term goal is to collect longitudinal observations from future projects for more comprehensive measurements). It remains to be seen how CoChaCo is perceived by knowledge engineers that do not want to develop modeling tools (see limitations commented below) or are accustomed to other method engineering approaches.

Qualities of the artifacts produced by the method. Its very nature suggests that CoChaCo was designed to benefit from the qualities of agile diagrammatic modeling – e.g., the possibility to customize interactive and dynamic notation, to capture variability through both graphical and model linking means and to easily implement Shneiderman's visualization mantra (overview first, zoom and filter, details on demand) [44]. For AMME practitioners CoChaCo fills a gap that used to be tackled through non-specific RE methods or improvised means - e.g., asemantic mind mapping lacking the streamlining and analysis support that a modeling method enables.

In the following, we summarize a managerial view on the qualities of the proposed artifact, including future work plans suggested in the *opportunities* sections:

Strengths: The proposed method repurposes the strengths of agile conceptual modeling for representing and managing modeling method requirements, thus addressing a specific gap in RE practice and literature. The proposal was driven by meta-requirements identified in method engineering projects and takes a hybrid KM / RE approach that may be further extended (CoChaCo itself evolves along the AMME lifecycle). The proposal may also be perceived as a more procedural and more structured way of mind mapping, aiming for a balance between loose note taking and conceptual modeling.

Weaknesses: Other methodologies that include conceptualization efforts (not only modeling tool development, but also language engineering or domain analysis) may benefit from it but they may have specificities that are not yet assimilated in CoChaCo. Evaluation should be extended over projects following related methodologies.

Opportunities: The provided support of the *Create* phase can be coupled with the subsequent phases through platform-specific plug-ins that take the conceptual structures represented with this method to an implementation format for a chosen metamodeling platform. Such an approach would enable the reuse of requirements represented in CoChaCo as a starting point for development. A plug-in for exporting directly into the ADOxx internal format is under development to demonstrate such streamlining.

Threats: CoChaCo adds an abstraction layer and enforces a method that RE practitioners are not typically familiar with, thus requiring a dedicated learning curve. Until now, CoChaCo was evaluated only with users that already had familiarity with the general operation of conceptual models. This is consistent with the fact that requirements captured with CoChaCo are supposed to drive AMME projects, therefore such expertise may be assumed to some extent.

However, an evaluation protocol should be devised for general purpose RE practitioners deciding to adopt CoChaCo for other purposes, where they should be enabled to leverage their own experience with general purpose RE frameworks.

REFERENCES

[1] I. Mirbel and J. Ralyte, "Situational method engineering: combining assembly-based and roadmap-based approaches" Requirements Engineering 11(1), pp. 58-78, 2005.

[2] U. Frank, "Domain-specific modelling languages: requirements analysis and design guidelines" in Domain Engineering, Springer, 2013, pp. 133–157.

[3] U. Frank, "Outline of a method for designing domainspecific modelling languages", ICB Research Reports 42, University Duisburg-Essen, Institute for Computer Science and Business Information Systems, 2010.

[4] D. Karagiannis, "Conceptual modelling methods: the AMME agile engineering approach", in Proceedings of Informatics in Economy 2016, LNBIP 273, Springer, 2017, pp. 3-19

[5] D. Karagiannis and H. Kühn, "Metamodelling Platforms", in Proceedings of EC-Web 2002 – DEXA 2002, LNCS 2455, Springer, 2002, pp. 182.

[6] R. A. Buchmann, A. Ghiran, C.C. Osman, and D. Karagiannis, "Streamlining semantics from requirements to implementation through agile mind mapping methods" in Proceedings of REFSQ 2018, LNCS 10753, Springer, 2018, pp. 335-351.

[7] A. Alfonso, V. Braberman, N. Kicillof, and A. Olivero, "Visual timed event scenarios". In Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), IEEE, 2004, pp. 168–177.

[8] BOC GmbH, The ADOxx metamodeling platform, http://www.adoxx.org/live/.

[9] D. Karagiannis, H. C. Mayr, and J. Mylopoulos (eds.), Domain-specific Conceptual Modelling, Springer, 2016.

[10] OMG: Meta-Modeling and the OMG Meta Object Facility, https://www.omg.org/ocup-2/documents/Meta-ModelingAndtheMOF.pdf.

[11] D. Karagiannis, R. A. Buchmann, P. Burzynski, U. Reimer, and M. Walch, "Fundamental Conceptual Modeling Languages in OMiLAB", in Domain-specific Conceptual Modelling, Springer, 2016, pp. 3-30.

[12] D. Karagiannis, R. Buchmann, and M. Walch, "How can diagrammatic conceptual modelling support knowledge management?", in Proceedings of ECIS 2017, Association for Information Systems, 2017, pp. 1568-1583.

[13] S. Kelly, K. Lyytinen, and M. Rossi, "MetaEdit+ a fully configurable multi-user and multi-tool CASE and CAME environment", in Seminal Contributions to Information Systems Engineering, Springer, 2013, pp. 109–129.

[14] H. G. Fill, T. Redmond, and D. Karagiannis, "Formalizing Meta Models with FDMM: the ADOxx Case", in Proceedings of ICEIS 2012, LNBIP 141, Springer, 2012, pp. 429-451.

[15] R. A. Buchmann and D. Karagiannis, "Agile Modelling Method Engineering: Lessons Learned in the ComVantage Project", in Proceedings of PoEM 2015, LNBIP 235, Springer, 2015, pp. 356-373.

[16] A. Awadid, D. Bork, and S. Nurcan, "Towards Assessing the Multi-view Modeling Capability of Enterprise Modeling Methods" in Proceedings of PoEM 2018, LNBIP 335, Springer, 2018, pp. 351-361.

[17] W. Maalej and A. K. Thurimella, Managing requirements knowledge, Springer, 2013.

[18] I. Nonaka, "The knowledge-creating company", Harvard Business Review 69, pp. 96-104, 1991.

[19] B. Nuseibeh, "Weaving together requirements and architectures". IEEE Soft., vol. 34, no. 3, pp. 115–117, 2001.

[20] OMiLAB. Bee-Up page in OMiLAB. http://austria.omilab.org/psm/content/bee-up/info.

[21] BABOK, The BABOK requirements taxonomy. 2019. https://blog.learningtree.com/the-babok-requirements-taxonomy/

[22] Requirements Solutions Group. 2008. A Requirements Taxonomy. https://businessanalysisexperts.com/Job_Aids_RSG/RequirementsTaxonomy.pdf

[23] D. Bork, R. Buchmann, D. Karagiannis, M. Lee, and E. T. Miron, "An Open Platform for Modeling Method Conceptualization: The OMiLAB Digital Ecosystem", Communications of the Association for Information Systems 44, pp. 673-697, 2019.

[24] OMiLAB, Open Models Initiative Laboratory – official website. http://omilab.org.

[25] D. Karagiannis, R. A. Buchmann, and D. Bork. "Managing Consistency in Multi-view Enterprise Models: an Approach based on Semantic Queries" in Proceedings of ECIS 2016, Association for Information Systems, paper 53, 2016.

[26] U. Frank, "Multi-Perspective Enterprise Modeling (MEMO) – Conceptual Framework and Modeling Languages", in Proceedings of HICSS-35, IEEE, 2002, pp. 1258-1267.

[27] R. Buchmann, M. Cinpoeru, A. Harkai, and D. Karagiannis. "Model-aware software engineering – a knowledge-based approach to model-driven engineering", in Proceedings of ENASE 2018, SciTe Press, 2018, pp.233-240.

[28] R. Petrusel, J. Mendling, and H. Reijers, "Task specific visual cues for improving process model understanding ", in Information and Software Technology 79, pp. 63-78, 2017.

[29] S. Strecker, U. Baumol, D. Karagiannis, A. Koschmider, M. Snoeck, and R. Zarnekow. "Five inspiring course (re-)designs", in Business & Information Systems Engineering, 61(2), pp. 241-252, 2019.

[30] J. E. Burge, Knowledge Elicitation Tool Classification, http://web.cs.wpi.edu/~jburge/thesis/kematrix.html#_Toc417957413

[31] P. Speel, A. Schreiber, W. van Joolingen, G. van Heijst, and G. Beijer. "Conceptual Modelling for Knowledge Based Systems", in Encyclopedia of Computer Science and Technology, Marcel Dekker Inc., 2001, pp. 107-132.

[32] W. M. P. van der Aalst, "Process-aware informations systems: lessons to be learned from process mining", in Transactions on Petri Nets and Other Models of Concurrency II, LNCS 5460, Springer, 2009, pp. 1-26

[33] B. Cheng and J. Atlee. "Research directions in requirements engineering", in Proceedings of ICSE 2007, IEEE, 2007, pp. 285–303.

[34] J. Ralyte. "Situational method engineering in practice: a case study in a small enterprise", in Proceedings of CAISE 2013 Forum, CEUR-WS 998, 2013, pp.17-24

[35] B. Henderson-Sellers, J. Ralyté, P. Âgerfalk, and M. Rossi, Situational Method Engineering, Springer, 2014.

[36] D. Gupta and N. Prakash, "Engineering methods from method requirements specifications". Requirements Engineering 6, pp. 135–160, 2001.

[37] D. L. Moody, P. Heymans, and R. Matulevičius, "Improving the effectiveness of visual representations in requirements engineering: An evaluation of i* visual syntax", in Proceedings

of 17th Requirements Engineering Conference, (RE 2009), IEEE, 2009, pp. 171-180.

[38] J. Beel, B. Gipp, and J. O. Stiller, "Information retrieval on mind maps-what could it be good for?", in Proceeding of the 5th International Conference on Collaborative Computing: Networking, Applications and Worksharing 2009, IEEE, 2009, pp. 1-4.

[39] F. Wanderley, D. Silveira, J. Araujo, A. Moreira, and E. Guerra, "Experimental evaluation of Conceptual Modelling through Mind Maps and Model Driven Engineering", in Proceedings of the 14th Interational Conference on Computational Science and Its Applications 2014. LNCS 8583, Springer, 2014, pp. 200-214.

[40] F. Wanderley and D. S. da Silveria, "A framework to diminish the gap between the business specialist and the software designer", in Proceedings of the 8th International Conference on Quality of Information and Communications Technology 2012, IEEE, 2012, pp. 199-204.

[41] D. Dermeval, J. Vilela, I. Bittencourt, J. Castro, S. Isotani, O. Brito, and A. Silva, "Applications of ontologies in requirements engineering: a systematic review of the literature". Requirements Engineering 21(4), pp. 405-437, 2016.

[42] K. Siegemund, E. J. Thomas, U. Aßmann, J. Pan, and Y. Zhao, "Towards ontology-driven requirements engineering", in Proceedings of the 7th Int. Workshop on Semantics-Enabled Software Engineering, 2011.

[43] L. Bass, J. Bergey, P. Clements, P. Merson, O. Ozkaya, and R. Sanghvan, "A comparison of requirements specification methods from a software architecture perspective", Technical report CMU/SEI-2006-TR-013, Carnegie Mellon University, 2006.

[44] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations", in Proceedings of IEEE Symposium on Visual Languages 1996. IEEE, 1996, pp. 336–343.