A Tool-Supported Methodology for Validation and Refinement of Early-Stage Domain Models

Marco Autili, Antonia Bertolino, Guglielmo De Angelis, Davide Di Ruscio, and Alessio Di Sandro

Abstract—Model-driven engineering (MDE) promotes automated model transformations along the entire development process. Guaranteeing the quality of early models is essential for a successful application of MDE techniques and related tool-supported model refinements. Do these models properly reflect the requirements elicited from the owners of the problem domain? Ultimately, this question needs to be asked to the domain experts. The problem is that a gap exists between the respective backgrounds of modeling experts and domain experts. MDE developers cannot show a model to the domain experts and simply ask them whether it is correct with respect to the requirements they had in mind. To facilitate their interaction and make such validation more systematic, we propose a methodology and a tool that derive a set of customizable questionnaires expressed in natural language from each model to be validated. Unexpected answers by domain experts help to identify those portions of the models requiring deeper attention. We illustrate the methodology and the current status of the developed tool MOTHIA, which can handle UML Use Case, Class, and Activity diagrams. We assess MOTHIA effectiveness in reducing the gap between domain and modeling experts, and in detecting modeling faults on the European Project CHOReOS.

Index Terms—Domain modeling, early stage model, model driven engineering, model refinement, model validation, natural language questionnaires, semantic model quality

1 Introduction

Models and abstractions are essential means in the development of modern complex software systems. A model offers an abstract representation of a system by focusing on concepts that are of relevance in a specific domain context and for a specific goal, while hiding away technical details that are out of scope for that context and goal.

In model-driven engineering (MDE) [1], models are the first-class artifacts along a system life-cycle, and the concrete system implementation is supported by systematic model refinement and automated transformations. MDE aims at reducing *the wide conceptual gap between the problem and the implementation domains* [2]. Following an MDE approach, developers can work close to the problem domain. Once they have specified a high-level model of the system to be developed, more detailed models can be obtained using model transformation tools. In the MDE vision, we can get rid of any potential problems of consistency or traceability between different projections or layers of a system design, as all transformations among the different artifacts will rely on verified technologies. Aiming at such a vision, research in MDE is very active, addressing challenges relative to modelling languages, separation of concerns, model evolution [3], manipulation and management [2], [4].

However, a weak link remains in the MDE chain of tool-supported model refinement: we are referring to the definition of the very first models, those that do not descend from another more abstract model by transformation. They need to be manually created from requirements already elicited from the interaction with the owners of the problem domain. We call this the *step zero* of model-driven development.

There is not much research in ensuring the validity of models built at step zero: how can we ensure that the manually created models properly meet the requirements elicited for the system to be built?

Today, the quality of conceptual models is still an immature notion whose definition is evolving [5]. A highly referenced quality framework is the one initially introduced in [6] and further extended in [7]. In this quality framework, different views of a model quality are introduced. Many existing model validation tools address the syntactic quality of a model, i.e., the correspondence between the model and the language in which the model is written. The above question refers to a different dimension for model quality, i.e., the relationship between the model and the domain that is modeled. This is called the semantic quality of a model. Semantic quality aims at validity, meaning that all elements in the model are correct and relevant for the domain, and at completeness, meaning that the model contains all the elements that would be correct and relevant about the domain.

Clearly, semantic quality is more difficult to ascertain than syntactic quality. Although some recent work [8], [9], [10] tries to mitigate this problem by allowing uncertainty to be expressed in early models, no tool could ever verify that a (syntactically correct) model properly represents all and only the concepts that are in the problem owner's mind.

- *M. Autili and D. Di Ruscio are with the Department of Information Engineering Computer Science and Mathematics University of L'Aquila, Italy. E-mail: {marco.autili, davide.diruscio}@univaq.it.*
- *A. Bertolino is with the CNR-ISTI of Pisa, Italy. E-mail: antonia.bertolino@isti.cnr.it.*
- *G. De Angelis is with the CNR-ISTI of Pisa, , Italy, and with the CNR-IASI of Rome, Italy. E-mail: guglielmo.deangelis iasi.cnr.it.*
- *A. Di Sandro is with the Department of Computer Science, University of Toronto, Toronto, ON M5S 2E4, Canada. E-mail: adisandro@cs.toronto.edu.*

Manuscript received 24 Apr. 2013; revised 6 Mar. 2015; accepted 31 May 2015. Date of publication 23 June 2015; date of current version 13 Jan. 2016. Recommended for acceptance by B.H.C. Cheng.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2015.2449319

Considering the step zero modeling activity, on the one hand there are the MDE developers who are the experts of modeling languages and tools; in this paper we refer to them as the Modeling Experts or MEs. On the other hand, there are the experts of the domain who know the system requirements very well, but are not necessarily experts of MDE approaches; we refer to them as the Domain Experts or DEs. As introduced in [11], to validate that the model rightly captures the intended domain knowledge, MEs and DEs need to mutually understand each other. The issue is that a gap in knowledge, background, and skills exists between MEs and DEs [2], [11]. In general, we cannot assume that the DEs yield the expertise required to directly inspect and navigate the models created by the MEs. Models can be convoluted notwithstanding the high abstraction level, especially for the representation of complex systems that involve different views and combinations of both static and behavioral models. Considering UML-based models, class diagrams can quickly scale up and become intractable for humans that are not experts of modeling notations. Visual aids and animation can be of help, but the ultimate means to validate that we are specifying the *right* model is to "ask the expert" [11]. The MEs talk to the DEs in natural language (NL) explaining what they are going to represent in the models, and collecting and processing DEs feedback. Tool support in this step can be extremely valuable, especially when models are large and span over many different conceptual domains.

Our work has been addressing the problem of validating the semantic quality of domain models for some time. Triggered by the need to validate a conceptual model in a real project, we have previously developed a prototype tool to help reduce the gap between MEs and DEs [11]. In its first version, our tool MOTHIA (standing for Model Testing by Human Interrogations & Answers) only processed UML *Class Diagrams*. Since then, with the purpose of validating multi-view system models, the tool has been extensively revised and extended in order to be able to handle two more diagrams, namely *Use Case Diagrams* and *Activity Diagrams*.

Our approach consists of the automatic derivation of a set of customizable questionnaires expressed in NL, from a model to be validated. Using such questionnaires, the MEs can interview the DEs and identify suspect portions of the models whenever the actual answer differs from the expected one. Notably, such approach aims at supporting MEs during their interaction with DEs. In this sense, the aim is to exploit the information in the artifacts to gather insights on some elements deserving to be carefully discussed, rather than to propose a framework as an alternative to the actual interaction among MEs and DEs.

The advantages of our approach are multiple. First, since we facilitate the interaction between the modelers and the problem owners, we get closer to the true semantic quality, in contrast with the *perceived semantic quality* as defined by Krogstie and Sølvberg [7]. They observed how any attempt to validate the semantic quality of a model always resolves to the comparison between two imperfect interpretations: on the one hand how the model is understood, and on the other the current knowledge of the domain, both of which could be wrong. This is always true: our approach mitigates the issue because it refers to MEs for interpreting the model and to DEs for drawing the domain knowledge, i.e., for each interpretation side, we consult the most competent stakeholder. Secondly, by using an inference engine that explores automatically and exhaustively all model elements according to some defined criteria, we make DEs interviews systematic. Finally, we can tune the questionnaires on specific portions of the model and regulate the number of questions we want.

The first results we reported in [11] about the application of a preliminary version of the tool MOTHIA in the Project IPERMOB[1] were encouraging. In this paper we repeat the assessment of the approach on a different, more extensive case study within the context of the European Project CHOReOS.[2] We agree with [12] that repeating the empirical assessment of research results is important. We aimed at verifying whether the approach confirmed its usefulness in revealing issues about the validity and completeness of conceptual models derived by MEs on behalf of the DEs. We also wanted to ascertain whether MOTHIA 's enhanced features improved the performance of the approach in terms of reducing the gap between MEs and DEs.

Summarizing, the contributions of this work include:

- a detailed description of the improved tool MOTHIA, enhancing the previous version in [11] with the capability to process Use Case and Activity Diagrams, and with a web-based engine to distribute the questionnaires and collect the answers;
- the application of MOTHIA to the domain model relative to the integrated development and runtime environment (IDRE) [13] delivered by the CHOReOS project;
- a repeated assessment of MOTHIA following the same process outlined in [11];
- a detailed comparison of the results in the two projects IPERMOB and CHOReOS.

The paper is structured as follows. We start by providing a brief overview of the methodology (Section 2). We continue with a more detailed description of the MOTHIA tool (Section 3), covering its architecture, our reference implementation and the Web distribution engine. We then present the application used for assessment, namely the CHOReOS IDRE conceptual model, along with the case study settings (Section 4), and discuss the obtained results (Section 5). We illustrate the process followed for model refinement, with examples of errors found (Section 6). We complete the assessment with a comparison between these results and those from our previous experience (Section 7), and a discussion of validity threats (Section 8). Related work is given in Section 9, and conclusions in Section 10 close the paper. An appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE-2013-04-0130, provides further raw data from the case study.

2 Overall Methodology

As explained in the introduction, this work stems from the realization that a knowledge gap often exists between DEs and MEs. Although DEs possess the domain knowledge needed to validate a candidate solution proposed by MEs,

1. http://www.ipermob.org/
2. http://www.choreos.eu/

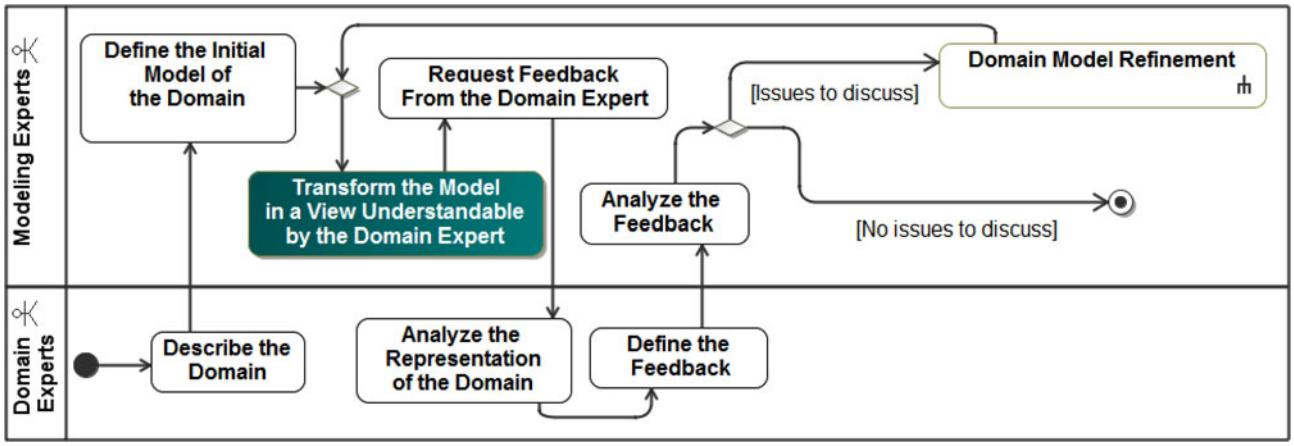


Fig. 1. Overview of a generic domain modeling process.

they can hardly do so, because the modeling artifacts are not easily comprehensible to them.

Thus, we introduce our approach and tool aiming at facilitating the mutual comprehension between DEs and MEs during the model validation phase. MEs remain free to pursue any strategy they deem appropriate in the interaction with DEs (e.g., arbitrarily asking about some model element). The results obtained from the application of MOTHIA can be used as an additional piece of information that MEs can exploit in such a strategy.

In the following, we overview our methodology for validation of model semantic quality that was originally proposed in [11]. The UML Activity Diagram in Fig. 1 models the steps involved in a domain modeling process, with an emphasis on the communications occurring between DEs and MEs.

A common practice in the definition of a domain model is to start from the specification of requirements already elicited for the domain to be modeled. In the hands of DEs, these requirements describe the domain (see the activity *Describe the Domain* in Fig. 1) and, as such, must be leveraged to constitute a common view of the domain. MEs can then propose an initial model of the domain, which needs to be validated by the DEs. To this purpose, an iterative process starts. Along the process iterations, the MEs try to make the model comprehensible to the DEs by generating and explaining appropriate model views. Based on such views, MEs request, collect, and analyze the feedback from the DEs to refine the initial model.

Although it is difficult to imagine these steps to be fully automated, MDE can support their implementation to achieve effective results in a more systematic way. With reference to Fig. 1, we support the automatic transformation of the model into views comprehensible by DEs (the dark-shaded activity in the figure). Precisely, we automatically generate a configurable list of simple questions expressed in NL, spanning over all the model elements. In principle, the questions just require a `Yes/No` answer; in practice (as we describe in Section 4.2), we also consider the case in which the DE is not able to answer, and provide more choices.

The idea is that at each iteration of the model validation process, MEs can interview the DEs by means of such automatically generated questionnaires. Thus, MEs are spared the difficult and cumbersome task of manually translating the model into NL descriptions. Furthermore, even though the capability of modelers should not be underestimated, this could also be an error-prone task due to potential ambiguities or biases in interpreting the semantics of the adopted modeling language (a good overview of threats is given in [14]).

Our approach relies on a formalization of the semantics of the modeling language the MEs chose. Thus, for each generated question it is possible to predict its expected answer by applying *semantics inference* on the current version of the domain model. A detailed description of the generation process of both the questions and their expected answers is reported in Section 3.

The subsequent steps concern the analysis of the feedback and their impact on the domain model. Specifically, if feedback reveal some mismatches between the domain model and the DEs intents (see the gateway after the activity *Analyze the Feedbacks*), the MEs should understand the nature of each mismatch and address it. We admit that in some cases it may be useful to validate the domain model by directly asking the DEs without the support of any additional information driving the discussion. However, we argue that our tool can help the MEs to focus the discussion on those parts of the model that contain issues and propose a new version of the domain model. This process can be iterated until all stakeholders are satisfied with the domain model. The resolution of the identified issues is performed in the *Domain Model Refinement* activity consisting of a number of steps, as we show in Fig. 7 and detail in Section 6.

3 Model Testing by Human Interrogations & Answers

In this section we describe first a generic architecture for the MOTHIA framework that supports the proposed methodology, and then the reference implementation for both the core framework and its web-based distribution engine.

3.1 The Architecture

As already introduced, MOTHIA deals with the knowledge gap that separates MEs and DEs: it takes a domain model as input and produces a set of NL questions as output. MOTHIA works on a formal logic representation of the input model, using an InferenceEngine to check for the satisfiability of desired properties. Fig. 2 depicts the internal structure of MOTHIA (left-hand side), how it interacts with

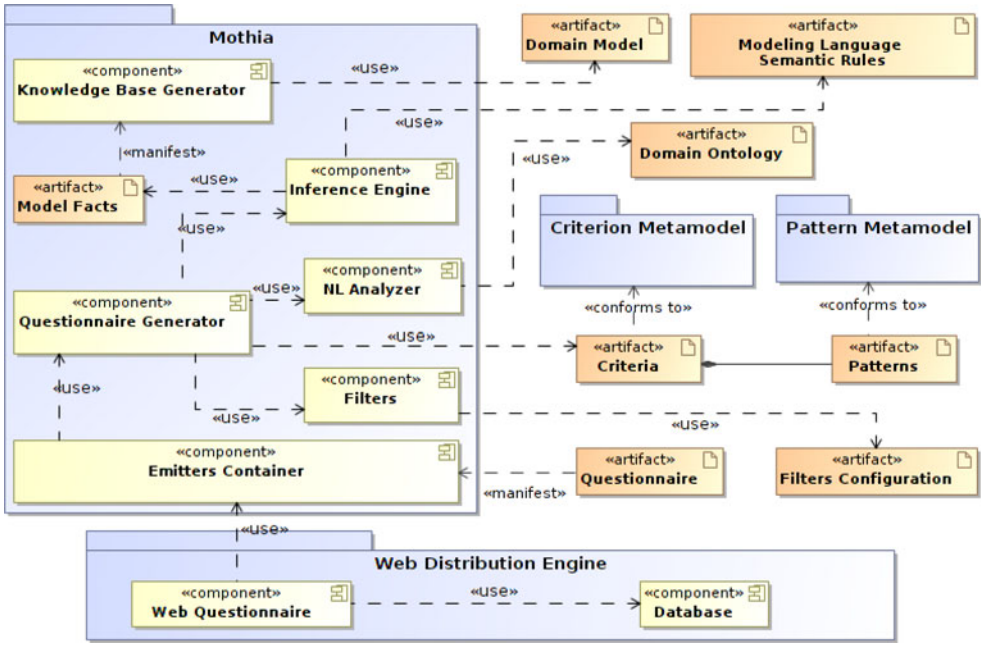


Fig. 2. The architecture of MOTHIA.

the model (right-hand side) and with the Web Distribution Engine (bottom). The KnowledgeBaseGenerator converts the input domain model into the internal representation, describing the domain entities and their relations as *facts*. The InferenceEngine loads such facts and the set of rules representing the semantics of the input modeling language. By querying the InferenceEngine, MOTHIA checks whether a given property is valid in the input domain model. Also, the InferenceEngine can return all the entities in the domain model satisfying a given property.

The QuestionnaireGenerator loads the configurations that drive the queries to the InferenceEngine and the creation of the questionnaire. The configurations of the QuestionnaireGenerator are expressed in term of *patterns* and *criteria*:

- the patterns represent syntactical combinations of elements in the input domain model that are deemed relevant, i.e., portions of the specific input domain model.
- the criteria define the properties that MOTHIA uses to explore and query the input domain model; in other words, a criterion abstracts a type of question in the questionnaire, the NL question template, and the strategy to compose patterns.

The NLAnalyzer aids the creation of NL questions by querying a domain *ontology*. It can either enrich and refine the text of a question, or try to infer entities and relationships that are not in the input model. Generated questions fall into three categories: a *Deduction* is inferred from `true` predicates on the input domain model, a *Distractor* from `false` predicates, a *Hypothesis* from predicates obtained through the domain ontology. From this classification we can argue that both Deductions and Distractors are means to validate the proper representation of the domain model (i.e., the validity goal of semantic quality [6]), while Hypotheses aim at validating its completeness (the second goal of semantic quality [6]).

Querying the input domain model from the set of criteria usually leads to a questionnaire with an intractably large number of questions. To reduce the number of questions, or even to focus only on specific areas of the domain model, a MOTHIA user can choose among several strategies. For example, experienced MEs could specialize the definitions of patterns and criteria and narrow the possible matching in the domain model. If such solution is not practical, the QuestionnaireGenerator uses the configurable Filters component. A filtering policy can be a sub-part selection of the input domain model, a random selection of questions, a word-based selection of questions or model elements, or a combination of them (other filterings can be devised).

Finally, the EmittersContainer component deals with the output format of the questionnaire. It can interact with a WebDistributionEngine in order to publish questionnaires on a web page and collect answers in a Database. This is especially useful in geographically distributed projects to ease and enhance cooperation among remote partners.

It is worth clarifying that the previous version of the tool presented in [11] only derived questionnaires for the validation of Class Diagrams, and only included Deductions and Distractors. All other features described below, including the derivation of Hypotheses for Class Diagrams, the validation of Use Case Diagrams, the validation of Activity Diagrams, as well as the Web-based Distribution Engine, constitute novel contributions of this work.

3.2 The Reference Implementation

MOTHIA is an Eclipse-based plugin, written using Java and EMF technologies [15]. MOTHIA can generate questions in NL from a variety of models: ECore diagrams, UML Class, Use Case and Activity diagrams. Even though MOTHIA supports multiple diagrams, the current version of the implementation does not handle cross-relationships among them. Extensions on this direction are planned as future work.

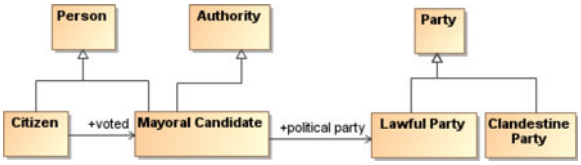


Fig. 3. The Class Diagram about an imaginary mayoral election.

According to the abstract architecture described in Section 3.1, the reference implementation uses SWI-Prolog as the InferenceEngine and WordNet [16] as the lexical ontology for the NLAnalyzer. The KnowledgeBaseGenerator currently handles a selection of the most relevant types of model elements and converts them to Prolog facts. As an example, the class `Mayoral Candidate` and its inheritance relationship from `Authority` in the Class Diagram of Fig. 3 are converted into the Prolog facts :i)

```
1) class('Mayoral Candidate', false)
2) generalization('Authority', 'Mayoral
     Candidate')
```

where: the first atom of the fact *1)* is the class name, the second a flag for its abstract modifier; the first atom of the fact *2)* is the superclass name, the second the subclass name.

In addition, the KnowledgeBaseGenerator can also codify sets of Prolog rules implementing the semantics of the supported input languages (e.g., the Liskov substitution principle for generalizations in both Class and Use Case diagrams). Thanks to this feature, arbitrary or domain-specific semantic interpretations of the model can be implemented (in Prolog) and plugged in this component.

The facts in the knowledge base are the building blocks to create patterns. As previously stated, a pattern is a syntactic structure in the input model. Facts are therefore composed together to form a pattern by means of a Prolog query. A pattern constructed this way can itself be used to incrementally compose subsequent patterns, allowing for arbitrarily complex yet manageable structures.

After patterns are identified in the model, criteria can use them to construct a set of NL questions. The criteria can be considered as the equivalent of a fault model in testing [17], hence the importance to identify a set of criteria that is as complete as possible. In our reference implementation, we defined criteria based on both the literature on model validation [18], [19] and on our own experience in modeling (e.g., [3], [20]).

The reference implementation, as well as the set of patterns and criteria we adopted, are available under an open-source licence.[3] To make the description concrete, in the following we discuss examples of patterns and criteria for each supported input model. These examples continue to be based on the already used model of an imaginary mayoral election.

3.2.1 Validating Class Diagrams

The reference implementation handles the following model elements: `uml:association`, `uml:class`, `uml:classAttribute`, `uml:dependency`, `uml:generalization`, `uml:package`. Table 1 reports the most relevant patterns defined in the reference implementation.

3. http://labsedc.isti.cnr.it/tools/mothia

Three noteworthy patterns apply to Fig. 3: the *marriage* pattern that matches classes `Person`, `Authority` (i.e., `super-class-A` and `super-class-B` respectively), `Citizen` (i.e., `sub-class-A`) and `Mayoral Candidate` (i.e., `sub-class-AB`); the *subclassesAssociation* pattern that matches classes `Authority`, `Party` (i.e., super-classes), `Mayoral Candidate` and `Lawful Party` (i.e., sub-classes); the *indirectAssociation* pattern that matches classes `Citizen`, `Mayoral Candidate` and `Lawful Party`, connected through the associations `voted` and `political party`.

A criterion can combine the three patterns together by means of a join operation (i.e., combining together the overlapping model elements). For example, let us define *Crit1c* as the criterion that looks for a class `C` that, according to the patterns in Table 1, is acting: as a sub-class in the pattern *subclassesAssociation*; as `sub-class-AB` in the pattern *marriage*; and also as an intermediate class in the pattern *indirectAssociation*. By querying the model in Fig. 3 with *Crit1c*, the class `C` would match with the class `Mayoral Candidate`; Deductions (marked with ✓) and Distractors (marked with ✘) can be derived from the associations of the matched class as follows:

- ✓ Can the relation from `Citizen` to `Mayoral Candidate` exist, where `Mayoral Candidate` is voted by `Citizen`?
- ✘ Can the relation from `Citizen` to `Lawful Party` exist, where `Lawful Party` is the political party of `Citizen`?

A Hypothesis (marked with **?**) may instead be built by looking for a relevant hyponym of (i.e., a kind-of) the noun `Party` in WordNet (e.g., in the following the bogus, and politically-correct, `FreeMonkeyIslandParty`), and creating a new class for the latter that inherits from the former:

- **?** Can the relation from `Mayoral Candidate`to `FreeMonkeyIslandParty` exist, where `FreeMonkeyIslandParty` is the political party of `Mayoral Candidate`?

Notably, the main difference in the above examples is that both the Deductions and Distractors use the entity `Lawful Party`, which was explicitly modeled within the domain model; on the contrary, in the case of the Hypotheses the concept `FreeMonkeyIslandParty` was not included at all. As described above, MOTHIA infers these new entities by querying an external source of information (i.e., WordNet) on some elements already represented in the domain model.

Further insights about the derivation of questions for the validation of Class Diagrams can be found in [11].

3.2.2 Validating Use Case Diagrams

The reference implementation for Use Case Diagrams handles the following model elements: `uml:actor`, `uml:association`, `uml:component`, `uml:package`, `uml:usecase`, `uml:extend`, `uml:include`, `uml:generalization`. Use Case diagrams may be coupled with some textual descriptions that clarify the functionalities or the services provided by the modeled system. Currently, MOTHIA only deals with the modeling elements represented within the diagrams alone; any corresponding textual description is currently discarded.

TABLE 1
Summary of Patterns for Class Diagram

Pattern	Description
attributeNotAssociation	matches class attributes that are not used as association member ends
chain	matches all sort of information about a class, to rank its relevance (attributes, associations from and to the class, multiplicities, parent and child classes)
family	matches a super-class (`super-class-A`) with sub-classes, some of which (`class-A`) have other sub-classes (`sub-class-A`)
indirectAssociation	matches all classes that are reachable from a certain class by navigating associations, with a configurable amount of intermediate classes
marriage	matches a class (`sub-class-AB`) that inherits from two super-classes (`super-class-A` and `super-class-B`), one of which has other sub-classes (`sub-class-AC`)
multiplicityRandom	generates a positive random integer between lower and upper bounds of an association multiplicity
siblings	matches sibling classes that inherit from the same super-class
subclassesAssociation	matches an association between two classes that are each derived from a distinct super-class
superclassesAssociation	matches an association between two classes that have each a distinct sub-class
unrelatedAssociation	matches two classes in an association relationship and a third class, with no association relationships to and/or from the previous two classes
unrelatedDependency	matches two classes in a dependency relationship and a third class, with no dependency relationships to and/or from the previous two classes
unrelatedInheritanceAssociation	matches two classes in a super/sub-class relationship and a third class, with no association relationships to the previous two classes
unrelatedInheritanceDependency	matches two classes in a super/sub-class relationship and a third class, with no dependency relationships to the previous two classes

The most relevant patterns defined in the reference implementation can be found in Table 2. It is interesting to note that some patterns are reused from the Class Diagrams, e.g., the *family*, *marriage*, *siblings* patterns. Whenever UML has abstractions across multiple diagram types (i.e., the `uml:generalization` in this case), a single pattern can be shared for all such diagrams.

Using the illustrative scenario of Fig. 4, in the following we will focus on the inclusion and extension relationships. These relationships are often misused or combined to form a complicated hierarchy, hence the need to stress them during the validation phase. For example, the *extensionInclusion* pattern can be found in Fig. 4, matching the use case `Vote Mayoral Candidate`, which extends `Vote` and is included by `Endorse Mayoral Candidate`.

Let us define a simple criterion *Crit1uc* that uses only the *extensionInclusion* pattern. Deductions can be derived from the inclusion and extension relationships of the matched use cases; Distractors can swap the semantics of these relationships:

- ✓ Is the use case `Vote Mayoral Candidate` a part of the use case `Endorse Mayoral Candidate`?
- ✓ Can the use case `Vote Mayoral Candidate` increment the behavior of the use case `Vote`?
- ✘ Can the use case `Endorse Mayoral Candidate` increment the behavior of the use case `Vote Mayoral Candidate`?
- ✘ Is the use case `Vote` a part of the use case `Endorse Mayoral Candidate`?

A Hypothesis can instead be made by looking for a synonym of the verb `Vote` in `Vote Mayoral Candidate` (`Choose` is the first synonym in WordNet), and referring to a new use case `Choose Mayoral Candidate` included by it:

TABLE 2
Summary of Patterns for Use Case Diagram

Pattern	Description
chain	matches all sort of information about an actor or a use case, to rank its relevance (associations from and to the node, super and sub generalizations, included and extended use cases)
extensionInclusion	matches a use case that extends a second use case and is included by a third use case
family	matches a super-entity (`super-usecase-A`) with sub-entities, some of which (`usecase-A`) have other sub-entities (`sub-usecase-A`)
inclusionExtension	matches a use case that includes a second use case and is extended by a third use case
marriage	matches an entity (`sub-usecase-AB`) that inherits from two super-entities (`super-usecase-A` and `super-usecase-B`), one of which has other sub-entities (`sub-usecase-AC`)
siblings	matches sibling entities that inherit from the same super-entity
subActorUsecaseAssociation	matches an association between an actor and a use case that inherit from a super-actor and a super-use case
superActorUsecaseAssociation	matches an association between an actor and a use case that have a sub-actor and a sub-use case
unrelatedActors	matches two unrelated actors
unrelatedAssociation	matches an actor and a use case with no associations
unrelatedUsecases	matches two unrelated use cases

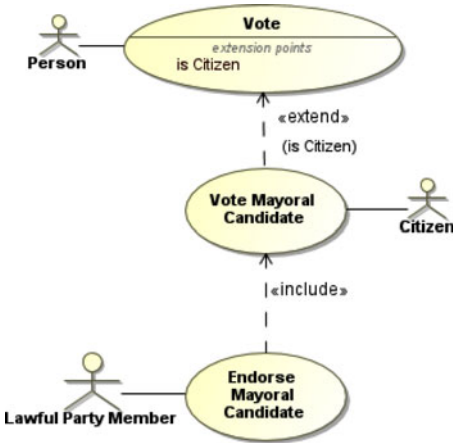


Fig. 4. The Use Case Diagram about an imaginary mayoral election.

? Is the use case `Choose Mayoral Candidate` a part of the use case `Vote Mayoral Candidate`?

A slightly more complex criterion *Crit2uc* can combine the previous *extensionInclusion* pattern with the `uml:association` fact. Note that such a combination is possible because a fact is the smallest possible pattern, since it strictly matches a model element. The elements that appear in both patterns will be joined together through Prolog unification. This means that a use case in this criterion will act both as a member of the *extensionInclusion* pattern and as the use case endpoint of the `uml:association` model element. Moreover, the actor endpoint of the `uml:association` model element becomes available, i.e., the actors `Person`, `Citizen` and `SomePartyMember` are associated with their respective use cases. For example, the following questions can be created:

✓ Can the actor `Lawful Party Member` get involved in the use case `Vote Mayoral Candidate` because it is part of the use case `Endorse Mayoral Candidate`?
✓ Can the actor `Person` optionally get involved in the use case `Vote Mayoral Candidate` instead of the use case `Vote`?
✘ Can the actor `Citizen` get involved in the use case `Vote Mayoral Candidate` because it is part of the use case `Endorse Mayoral Candidate`?
✘ Can the actor `Citizen` optionally get involved in the use case `Vote Mayoral Candidate` instead of the use case `Vote`?

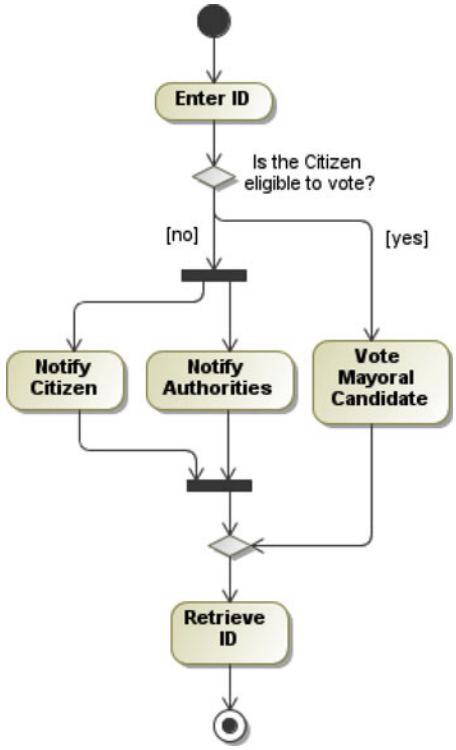


Fig. 5. The Activity Diagram about an imaginary mayoral election.

? Can the actor `Citizen` get involved in the use case `Choose Mayoral Candidate` because it is part of the use case `Vote Mayoral Candidate`?

3.2.3 Validating Activity Diagrams

The reference implementation for the Activity Diagrams handles the following model elements: `uml:action`, `uml:decision`, `uml:event`, `uml:flow`, `uml:fork`, `uml:join`, `uml:merge`, `uml:object`, `uml:signal`. The most relevant patterns defined in the reference implementation can be found in Table 3.

The illustrative scenario of Fig. 5 details the use case `Vote Mayoral Candidate` of Fig. 4 for an electronic voting machine. In this scenario three noteworthy patterns can be found: the *alternativeActions* pattern that matches with both the set {`Vote Mayoral Candidate`, `Notify Authorities`}, and the set {`Vote Mayoral Candidate`, `Notify Citizen`}; the *concurrentActions* pattern that

TABLE 3
Summary of Patterns for Activity Diagram

Pattern	Description
alternativeActions	matches two alternative actions
alternativeDecision	matches the alternative guards of a decision
chain	matches all sort of information about a node, to rank its relevance (type, flows from and to the node, signals, events and loops in which the node can be involved)
concurrentActions	matches two concurrent actions
graphFlow	matches a flow based on multiple node types
graphLoopFlow	matches a loop flow based on multiple node types
graphShortFlow	matches a short flow with regards to intermediate node types
notInFinalFlow	matches a node that is not in a flow to the final node
notInFullFlow	matches a node that is not in a flow from the initial node to the final node
notInInitialFlow	matches a node that is not in a flow from the initial node
unrelatedActions	matches an action with no flows to and from another action

matches with the set {`Notify Authorities`, `Notify Citizen`}; the *graphFlow* pattern that can match with any flow in the activity graph from a node to another one, where each node in a flow matches with a given set of node types.

Specifically, let us consider the *graphFlow* pattern implemented as a Prolog predicate with the following signature:

```
graphFlow(NodeTypesSet, StartNode,
EndNode, IntermediateNodesSet)
```

By specifying one or both the endpoints (i.e., `StartNode` and `EndNode`), the pattern will narrow down the results making the predicate true or possibly returning an empty set. The flows matched by the pattern are equivalent to graph paths e.g., no attempt is made to address the scheduling of actions in a concurrent section. For example, by specifying the model element `uml:action` as first parameter, and the starting and final activity nodes as second and third parameters, respectively, the *graphFlow* pattern will match the following ordered sets:

- { `Enter ID`, `Vote Mayoral Candidate`, `Retrieve ID`}
- { `Enter ID`, `Notify Authorities`, `Retrieve ID`}
- { `Enter ID`, `Notify Citizen`, `Retrieve ID`}

In the same example, by changing the third parameter to be the node `Retrieve ID`, the *graphFlow* pattern would match the following ordered sets:

- { `Enter ID`, `Notify Citizen`}
- { `Enter ID`, `Notify Authorities`}
- { `Enter ID`, `Vote Mayoral Candidate`}

The current implementation of MOTHIA also defines querying criteria on the Activity Diagrams, inspired by Temporal Logic operators and well-known qualitative patterns for property specifications in finite-state systems [21]. NL questions are loosely based on successive work on structured English grammar for such properties in real-time systems [22]. Specifically, the criteria have been implemented by referring to the *Next*, *Future*, *Always*, and *All* operators. Each criterion has been defined by composing together the *graphFlow* pattern and some first order predicates.

Deductions can be created by using the actions that are matched, Distractors instead by using any combination of the unmatched actions. Similarly to the example given in Section 3.2.2, the Hypotheses can be created by introducing a new action `Choose Mayoral Candidate` preceding `Vote Mayoral Candidate`. In the following, each criterion is formally defined and a number of example questions are shown.

Let $\mathcal{A}(ad) = \{a_1, \ldots, a_n\} \cup \{a_I\} \cup \{a_F\}$ be the set of actions (i.e., activities) of an activity diagram ad, where a_I is the activity initial node, and a_F the activity final node.

The *Next* criterion creates questions about strictly consecutive actions and is defined as:

$$Next(a_i) = a_j \Longleftrightarrow \texttt{graphFlow}(\{\texttt{uml : action}\}, a_i, a_j, \emptyset) = true. \quad (1)$$

Note that, by specifying an empty intermediate set as fourth parameter, the `graphFlow` predicate is true if, in the activity diagram ad, no other action is specified in between a_i and a_j. Applying this criterion against the scenario in Fig. 5 generates, among the others, the following questions:

- ✓ Can `Vote Mayoral Candidate` be the next action to be executed after the action `Enter ID`?
- ✘ Could `Enter ID` be the previous action executed before the action `Retrieve ID`?
- ? Can `Vote Mayoral Candidate` be the next action to be executed after the action `Choose Mayoral Candidate`?

Extending the notion of *Next* criterion, the *Future* criterion creates questions about actions that are not necessarily (strictly) consecutive. Thus, let $\mathcal{T}(ad)$ be the set of totally ordered subsets of $\mathcal{A}(ad)$ representing all the possible (sub) traces of actions, as ordered according to the action flows imposed by the activity diagram ad. Each trace $t_{i,j} = \{a_i, \ldots, a_j\} \in \mathcal{T}(ad)$ is such that $t_{i,j}[k{+}1] = Next(t_{i,j}[k])$, $\forall\, 0 \leq k < |t_{i,j}|$, where $t_{i,j}[k]$ is the k-th action of the trace $t_{i,j}$.

Given a trace $t_{i,j}$, another trace $t^{in}_{i,j}$ is said to be a *strict inner trace* of $t_{i,j}$, denoted as $t^{in}_{i,j} \prec t_{i,j}$, iff:

- either $t^{in}_{i,j} = \emptyset$ and $t_{i,j} = \{a_i, a_j\}$,
- or $t^{in}_{i,j} = \{a_k\}$ and $t_{i,j} = \{a_i, a_k, a_j\}$,
- or $t^{in}_{i,j} = \{a_p, \ldots, a_q\}$ and $t_{i,j} = \{a_i, a_p, \ldots, a_q, a_j\}$.

With these premises, for a given activity diagram ad, the *Future* criterion is defined as:

$$Future(a_i) = a_j \Longleftrightarrow \exists t^{in}_{i,j} \prec t_{i,j} \in \mathcal{T}(ad) | \texttt{graphFlow}(\{\texttt{uml : action}\}, a_i, a_j, t^{in}_{i,j}) = true. \quad (2)$$

As already introduced above, the *Next* criterion is a special case of the *Future* criterion, where $t^{in}_{i,j} = \emptyset$. Applying the *Future* criterion against the scenario generates, among the others, the following questions:

- ✓ Can the action `Retrieve ID` be executed after the action `Enter ID`?
- ✘ Could the action `Notify Citizen` be executed before the action `Notify Authorities`?
- ? Could the action `Choose Mayoral Candidate` be executed before the action `Retrieve ID`?

Extending the notion of *Future* criterion, the *Always* criterion creates questions about actions that, through all possible flows, must be necessarily followed by a specified action in the future. Given an activity diagram ad, the *Always* criterion is defined as:

$$Always(a_i) = a_j \Longleftrightarrow \forall t^{in}_{i,F} \prec t_{i,F} \in \mathcal{T}(ad), a_j \in t^{in}_{i,F} \wedge \texttt{graphFlow}(\{\texttt{uml : action}\}, a_i, a_F, t^{in}_{i,F}) = true \quad (3)$$

Note that $t_{i,F} = \{a_i, , a_F\}$ represents a trace that goes from the action a_i to the activity final node a_F. Applying the *Always* criterion against the scenario generates, among the others, the following questions:

- ✓ Is the action `Retrieve ID` always executed after the action `Enter ID`?
- ✓ Is `Enter ID` always the previous action executed before the action `Vote Mayoral Candidate`?
- ✘ Is the action `Vote Mayoral Candidate` always executed before the action `Retrieve ID`?
- ✘ Is `Notify Authorities` always the next action to be executed after the action `Enter ID`?
- ? Is `Choose Mayoral Candidate` always the previous action executed before the action `Vote Mayoral Candidate`?

The *All* criterion creates questions about actions that must occur in all possible traces from the activity initial node a_I to the activity final node a_F. It is defined as:

$$All(a_i) = true \Longleftrightarrow \forall t^{in}_{I,F} \prec t_{I,F} \in \mathcal{T}(ad), a_i \in t^{in}_{I,F} \\ \wedge \texttt{graphFlow}(\{\texttt{uml : action}\}, a_I, a_F, t^{in}_{I,F}) = true. \quad (4)$$

Applying this criterion against the scenario cannot generate any Deduction; it generates, among others, the following Distractor:

✘ Is the action `Notify Authorities` always executed?

For the sake of completeness we point out that none of these patterns has been reused from Class or Use Case Diagrams. In fact, the syntactical definition of Activity Diagrams reflects their behavioral nature, making it harder to benefit from the reuse mechanism outlined at the beginning of Section 3.2.2. This would change if, for example, MOTHIA were to support State Machine or Sequence Diagrams.

3.3 The Web-Based Distribution Engine

Another new feature offered by MOTHIA is abQuestionnaire,[4] a web-based engine supporting the distribution of questionnaires to the DEs and the collection of their feedbacks. This engine is the reference implementation of the abstract component WebDistributionEngine, presented in Section 3.1.

In practice, this component becomes an important asset for our approach when the people involved in the modeling activity belong to different organizations, and limited face-to-face interactions can be organized (this was the case in the context of the European Project CHOReOS in which we applied the approach).

Specifically, abQuestionnaire is a plugin for Wordpress.[5] Like most web-based applications, it is structured into two main components: a back end and a front end.

In the back-end component, registered users can administrate the questionnaires generated by MOTHIA. The plugin provides canonical functionalities such as uploading, publishing, and unpublishing a questionnaire. In addition, it also gives support to invite the interviewed persons, and to report useful statistics about the published questionnaires.

After the publishing phase, the front end of abQuestionnaire allows invited persons (i.e., the DEs) to fill in their questionnaires. Fig. 6 shows a screen-shot of the abQuestionnaire's front end on a generated questionnaire.

4 Case Study

This section reports our experience in applying MOTHIA in the context of the CHOReOS project. The case study was structured to deal with two different aspects: validate and refine the CHOReOS domain model; evaluate the performance of MOTHIA in reducing the gap between MEs and DEs, when used to support the former activities. As detailed in the following, DEs were only aware of collaborating to the application of the methodology in Fig. 1; but actually they were involved in two different experiences that could not be distinguished by them.

4. http://labsedc.isti.cnr.it/tools/abquestionnaire
5. http://wordpress.com/

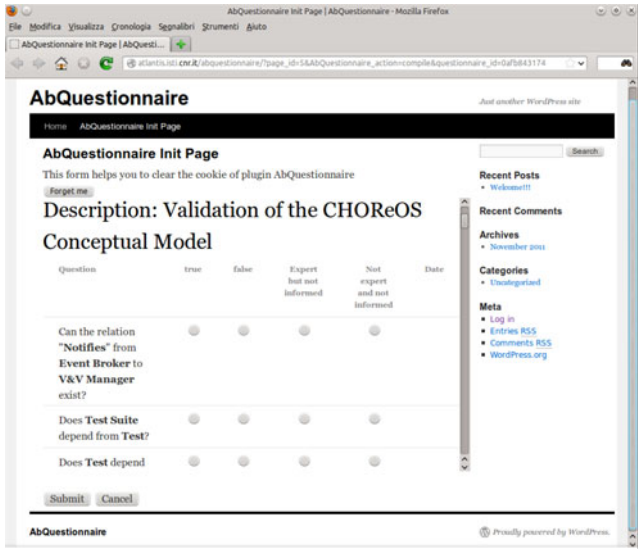


Fig. 6. abQuestionnaire.

It is worth to clarify that our experimental setting differs from a "real-life" scenario due to many aspects and constraints that are discussed in Section 8. Among the others: the selection of the DEs, the size of the questionnaires, the absence of a reference baseline and the way the authors compensated it.

In the following, we first introduce the reference scenario. Then, we report the case study settings and the conducted activities.

4.1 Scenario

The CHOReOS project investigates a set of core methodologies and tools for the development of large-scale distributed applications obtained from the loose interaction among independent third-party services. The project binds such type of applications to the notion of a *service choreography* [23], defined as the high-level specification of the desired (functional and non-functional) interaction protocol, but without a concrete refinement of services implementation and internals. Precisely, the CHOReOS project targets scalable solutions for the ultra-large scale challenges of Future Internet [24] choreographies taking into account size, distribution, heterogeneity, and dynamism [25].

From a technical point of view CHOReOS has developed and promoted a model-driven development process of choreographies, and a supporting framework referred to as the CHOReOS IDRE [13]. Specifically, the CHOReOS IDRE relies on the integration, interoperability, and large scale distribution capabilities provided on the Enterprise Service Bus middleware paradigm. Such a paradigm has been enhanced in order to cope with the heterogeneous interaction semantics and the sophisticated service discovery mechanisms; it has been also configured to deal with assets ensuring the quality of services and choreographies through the governance and the verification & validation (V&V) approaches at both design and run-time [26], [27].

Independently from the above hinted implementation and technological details, the CHOReOS project has developed a specific domain model (also referred to as the CHOReOS *conceptual model*) abstracting the facilities offered by the IDRE.

The version of the CHOReOS IDRE domain model preceding its validation was specified by means of 22 diagrams, counting: a) 62 classes, 89 associations, four dependency relationships, 16 inheritance relationships in four Class Diagrams; b) 14 actors, 28 use cases, 38 relationships in four Use Case Diagrams; c) 105 object and activity nodes, 38 control nodes, 166 flows in 14 Activity Diagrams. The final release of the CHOReOS Conceptual Model is accessible through the official project web-site.[6]

4.2 Case Study Design and Collected Answers

We used MOTHIA to generate 18 different questionnaires. This number corresponds to the number of DEs from partners having both expertise and effort for validating the CHOReOS domain model on behalf of the whole project. Due to the size of the consortium, the criteria adopted to select the DEs were mainly based on requesting the partners to candidate some reference person in their organization/institution competent in designing, developing, validating, or governing distributed and service oriented applications.

Each questionnaire contained 30 questions, spanning over all the areas of concern in the domain model. The number of questions was fixed so to keep the estimated time for filling in a questionnaire under one hour. The questions included both Deductions and Distractors.

Note that, even though the current version of MOTHIA also supports the generation of Hypotheses (see Section 3), we decided not to use this feature in the case study. In fact, including Hypotheses would refer to the assessment of ontological aspects in validating semantic quality that by itself would deserve a dedicated study. We deemed that even without considering Hypotheses, the case study was already dense enough with novel aspects to be evaluated. Nevertheless, Section 6 speculates on how the adoption of Hypotheses can support the detection of some of the most common errors that the domain model could include.

As described in Section 2, MOTHIA produces NL questions ideally requiring a `Yes/No` answer. In practice, the questions actually propose a multiple-choice answer based on four possible, and exclusive, cases: (1) `Yes`, (2) `No`, (3) "I'm not an expert of this specific subject" (`NE`), and (4) "I'm an expert of the subject but I'm not sure what to answer" (in brief, `DK` for don't know).

The rationale behind the four choices is as follows. Answers (1) and (2) constitute the basic choices: through them we can assess a concept that a DE would like to formalize in the domain model (i.e., its properties, relations with other concepts, etc) against its actual representation in the model. Option (3) has been introduced because, especially for large models, a DE may not be knowledgeable of all features. Finally, option (4) is quite useful for dealing with questions whose formulation can be confusing, like Distractors (we recall from Section 3.1 that a Distractor is created from false predicates on the input model, e.g., a non-existent relationship among two or more entities). We admit that an expert could also answer `DK` because the question is not clear: in this case effort would be wasted in the post-analysis of the model (see Section 6). Nevertheless, in general we noticed that an answer (4) can stimulate the discussion on aspects that were not directly included in the domain model.

At the time the questionnaires were generated, the errors included in the input domain model were obviously unknown. In addition, no reference baseline of the CHOReOS domain model was available in order to compare the errors discovered by using MOTHIA. Therefore, we were not able to formulate any performance indicator measuring the percentage of errors revealed by MOTHIA.

The absence of a reference baseline could have been compensated by creating two separate working groups: one where DEs and MEs interact without any specific guidance; the other where MEs plan the discussion with DEs focusing on those portions of the domain model that MOTHIA suggested as including potential issues.

On the one hand, MOTHIA 's performance indicators could have been estimated by comparing the results obtained from both these working groups. On the other hand, such comparative analysis would not have been able to estimate the percentage of the detected errors, as the total number of errors in the domain model would have still been unknown. Furthermore, the overall commitment from the CHOReOS project was to validate the domain model rather than assess a novel validation approach. The setup of the experience had to deal with the limitations in both budget and available professionals always imposed in any projects (in our case, only 18 DEs, and few MEs).

Based on these observations, we deemed improper to split the available resources in two separate working groups. As in [11], we exploited the idea to submit to each DE a single questionnaire targeting two different experiences: a first set of questions aiming at validating the CHOReOS domain model (on average 25 questions), a second set of questions aiming at evaluating the fault detection capability of MOTHIA (on average five questions).

In [11], we found it useful to adopt an approach similar to that of mutation testing [17], [28], which creates mutants (i.e., slightly different versions) of source artifacts by injecting artificial faults. In such a way, the total number of injected faults is known, so it is possible to calculate the percentage of mutants that are identified during the validation step.

In a similar way to mutation testing, we modified the domain model by injecting random, yet controlled faults. Consequently, we wanted to obtain an estimation of how good is MOTHIA in detecting real (and unknown) faults that the domain model may contain by observing how good it is in detecting the injected artificial faults.

However, note that for the already mentioned reasons of limitedness of resources, we could not faithfully follow the mutation testing paradigm in our evaluation of MOTHIA capability to detect injected faults. In software mutation testing, a program is subject to a systematic mutation process in all and every possible code element that could be mutated in several different ways (ending up with producing an experimental basis of typically thousands or hundred thousands mutants). Then, a test suite generated according to a given test technique is executed on the golden version and on *all* mutants, and the fault detection capability of the testing technique is assessed by the proportion of killed mutants.

In our study, where the "testing technique" under evaluation is MOTHIA and the "test subject" is a model, to

6. http://www.choreos.eu/bin/Download/Deliverables

TABLE 4
Kinds of Generic Faults Mutating the Diagrams

Fault Id	Description
F1	Add Edge
F2	Change Edge Endpoint
F3	Change Edge Direction
F4	Change Edge Type/Annotation
F5	Delete Edge

faithfully follow the mutation paradigm would have required first to perform a systematic mutation of all elements in the model, second to apply MOTHIA on each obtained mutant to derive a questionnaire and finally that every questionnaire from each mutated model is answered by an expert. Doing so was clearly beyond our available resources: as said, we could only ask 18 DEs. Speaking in terms of mutation testing, our situation was like if given the many (thousands of) mutants that could be obtained from a program, we could only run the test suite on a very few (18) of them.

Therefore, in a similar way to mutation testing we inserted some random faults, but after this the similarity becomes looser: to be certain to get some data on fault detection, we ensured that each of the 18 DEs was exposed to a few questions derived from some mutated models (in addition to those questions truly aimed at validating the model). More precisely, for the purpose of this case study, MOTHIA was used on both the mutated models and the original models. Then, each questionnaire was configured to contain a mix of questions affected by the artificial faults (i.e., *faulty questions*) and questions not involved in the mutation process (i.e., *genuine questions*). The mean number of faulty questions per questionnaire was set to 5, using a filter to randomly choose from those generated on the mutated models (see Section 5.1 for further details).

In this way, what we can assess through the faulty questions we introduced is the likeliness that a fault in a certain element of a model is identified *given that* a question about that element is asked to a DE. Due to lack of resources (i.e., DEs) we could *not* also assess the effectiveness of MOTHIA concerning the likeliness that such a faulty element would be hit when generating a questionnaire.

The definition of proper mutation operators is a critical task. We based this activity on examples discussed in the literature about mutation testing for both models and model transformations [28]. We also abstracted this process to make the mutation operators applicable to generic graphs, i.e., to any of the evaluated diagrams. Specifically, in the CHOReOS domain model we applied five types of mutations that are reported in Table 4.

The introduced mutations only affect the edges contained in the diagrams (i.e., either relations, or flows). In fact, any modification of the entities in the diagrams (i.e., classes, actors, use-cases, actions) was not considered as an interesting mutation. In particular, deleting any entity would only imply that no question will cover it, and the modification would be invisible. Similarly, the inclusion of a new entity, which has no relations (i.e., navigable edges in the graph the diagram subsumes) with the rest of the model elements, would not match any realistic pattern. Finally, the modification of the name (i.e., a label) of an entity would concern ontological aspects that, as already said, are not covered by this case study.

We notice once again that the mutation process was introduced with the purpose of evaluating MOTHIA and not the domain model. Thus, on the one hand, the interviewees were ignoring that some of the questions they answered were subject to such adulteration; on the other hand, during the post-processing of the questionnaires, we actually pulled apart the questions originated from any of the injected faults and processed them separately as a different study.

The answers we collected for genuine and faulty questions are summarized in Tables 5 a and in 5 b, respectively. Each row refers to a questionnaire (and hence to a participant DE).

As already said, in this case study we refer to a question as to a test on the input model; with reference to Table 5 a, a test is said to `Pass` if the answer of the DE matches the answer expected by MOTHIA ; to `Fail` otherwise.

In mutation testing, a mutant is said to be "killed" when its outcome on a test case is different from the outcome of the original program on the same test case [29]. Similarly, here the notion of killing a mutant corresponds to the identification of an introduced fault. A model mutation is "killed" (for brevity `K`, see Table 5 b) when the answer to one of the faulty questions affected by it is different from the answer inferred by MOTHIA (i.e., `UN`-expected column in Table 5 b), or if the answer is `DK`. The reason why we marked a `DK`-answered faulty question as killed stems from the following reasoning: for any question on which a DE declares him (her)self as an expert and not able to express an opinion at the same time, it is likely that the MEs will want to look closer to those model elements that originated the question. The tool might have in fact generated a confusing question due to the presence of a model error (an injected fault in this case). The consequent analysis on those elements will likely lead to the discovery of the problem.

5 Evaluation of the Results

This section evaluates the results collected in the case study.

It is worth considering that our experience refers to a real European project, and not a simulated study. The partners involved in the use of MOTHIA were contributing on a voluntary basis. Certainly, their participation was helpful and valuable. Nevertheless, as a well-known issue in empirical software engineering [12], we experienced the difficulties and costs of experimenting with professionals within development projects. Indeed, the results we collected should be considered as a valuable step towards dealing with the research questions we are trying to address, rather than the "final proof". A critical discussion about the factors that may threaten the validity of our conclusions are presented separately in Section 8.

As described in Section 4.2, we submitted to each interviewee a questionnaire including questions that target two separate studies: measuring the ability of MOTHIA questionnaires to reveal errors, and assessing if MOTHIA reduced the knowledge gap between DEs and MEs. The former study is presented in Section 5.1, the latter in Section 5.2.

TABLE 5
The Experts' Answers

(a) Results for the genuine questions.

Questionnaire	#Pass	#Fail	#NE	#DK
PART_1	12	9	3	3
PART_2	14	5	2	4
PART_3	13	5	5	0
PART_4	11	3	10	0
PART_5	17	6	0	0
PART_6	11	2	0	12
PART_7	14	7	0	7
PART_8	13	6	6	0
PART_9	20	4	3	0
PART_10	16	9	1	0
PART_11	19	6	0	0
PART_12	7	3	14	0
PART_13	17	4	0	4
PART_14	18	5	0	0
PART_15	8	8	11	0
PART_16	19	3	2	0
PART_17	3	5	17	0
PART_18	2	4	20	0
TOTALS	234	94	94	30

(b) Results for the faulty questions.

Questionnaire	# Faulty Questions	#UN	#DK	#K (#UN+#DK)	#NK	#NE
PART_1	3	2	0	2	0	1
PART_2	5	0	0	0	4	1
PART_3	7	3	0	3	1	3
PART_4	6	2	0	2	2	2
PART_5	7	6	0	6	1	0
PART_6	5	2	1	3	2	0
PART_7	2	1	1	2	0	0
PART_8	5	0	0	0	2	3
PART_9	3	2	0	2	0	1
PART_10	4	1	0	1	3	0
PART_11	5	3	0	3	2	0
PART_12	6	2	0	2	1	3
PART_13	5	3	1	4	1	0
PART_14	7	3	0	3	4	0
PART_15	3	0	0	0	0	3
PART_16	6	2	0	2	1	3
PART_17	5	1	0	1	0	4
PART_18	4	2	0	2	1	1
TOTALS	88	35	3	38	25	25

K	The mutant is Killed.
NK	The mutant is Not Killed.
UN	Unexpected – the DE gave an answer different from MOTHIA.
DK	Don't Know – the DE missed some information.
NE	The DE is Not Expert of this topic.

TABLE 6
Comparison between Faulty Questions by Diagrams

Questionnaire	Class Diagrams			Activity and Use Case Diagrams		
	# Faulty Questions	#K	#NE	# Faulty Questions	#K	#NE
PART_1	2	1	1	1	1	0
PART_2	3	0	0	2	0	1
PART_3	2	2	0	5	1	3
PART_4	4	1	1	2	1	1
PART_5	2	2	0	5	4	0
PART_6	1	0	0	4	3	0
PART_7	2	2	0	0	0	0
PART_8	4	0	3	1	0	0
PART_9	0	0	0	3	2	1
PART_10	2	0	0	2	1	0
PART_11	4	2	0	1	1	0
PART_12	0	0	0	6	2	3
PART_13	4	3	0	1	1	0
PART_14	3	0	0	4	3	0
PART_15	1	0	1	2	0	2
PART_16	0	0	0	6	2	3
PART_17	4	1	3	1	0	1
PART_18	1	0	1	3	2	0
TOTALS	39	14	10	49	24	15

K	The mutant is Killed.
NE	The DE is Not Expert of this topic.

5.1 Can MOTHIA Questionnaires Help Detect Modeling errors?

Section 4.2 described the settings of the study aimed at assessing the ability of MOTHIA to detect modeling faults based on mutations of the original domain model. Specifically, with respect to the type of faults presented in Table 4, we randomly mutated a single element of each diagram of the CHOReOS domain model. These mutations in turn generated 88 faulty questions. Please refer to Table 5 b for a detailed breakdown, and to dimension D7.1 in Section 7 for a detailed analysis of the rationale behind such numbers.

A first notable result is that all five fault types described in Table 4 were detected at least once. Furthermore, the *ex-post* analysis revealed that more than half of the faulty questions were killed (i.e., 60.32 percent if we exclude those questions in which the DEs declared themselves not expert), allowing the detection of almost all the introduced faults (i.e., 16 faults found of the 19 considered in total, or 84.21 percent). The detailed presentation of these results can be found in Section 7, dimension D7.1.

Evidence that MOTHIA helps in detecting model errors was also collected in the second study. As reported in the column Fail in Table 5 a, the DEs collected several answers from the MEs that differed from the ones expected by MOTHIA. The discussion stimulated by such conflicts allowed for the detection of real errors that had slipped into the model, as more deeply described in Section 6.

In addition, we compared the faulty questions that had been generated and killed from the Class Diagrams with the ones from both the Activity and Use Case Diagrams (i.e., the new features of MOTHIA). The data relative to this case study shows that, excluding questions for which the MEs declared themselves not expert, the new features scored a better result with respect to the faulty questions killed (i.e., $\frac{24}{49-15} = 70.59\%$ against $\frac{14}{39-10} = 48.27\%$ – see Table 6).

5.2 Does MOTHIA Help to Reduce the Gap between DEs an MEs?

In order to assess how our approach supports the interaction between DEs and MEs, we formulated two metrics to estimate the *efficacy* and the *adequacy* of the questionnaires generated with MOTHIA.

As explained in Section 4.2, the questionnaire answers can be divided into those classifying the interviewee as an "expert" of the specific topic tackled by the question (i.e., Yes, No, DK), and those classifying him/her as "not expert" (i.e., NE).

To assess *efficacy*, we defined a metric evaluating the number of questions on which the interviewees were able to express an opinion (i.e., answer Yes or No), over the total number of questions for which they classified themselves as "experts" of the topic:

$$\frac{\#Pass + \#Fail}{\#Total - \#NE} \times 100. \quad (5)$$

The metric in Equation (5) estimates the *efficacy* of MOTHIA in formulating questions as it reflects the number

TABLE 7
Efficacy per Questionnaire

Questionnaire	Efficacy	Efficacy on Class Diagrams	Efficacy on Activity and Use Case Diagrams
PART_1	87.50%	88.89%	86.67%
PART_2	82.61%	90.00%	76.92%
PART_3	100.00%	100.00%	100.00%
PART_4	100.00%	100.00%	100.00%
PART_5	100.00%	100.00%	100.00%
PART_6	52.00%	58.33%	46.15%
PART_7	75.00%	88.89%	68.42%
PART_8	100.00%	100.00%	100.00%
PART_9	100.00%	100.00%	100.00%
PART_10	100.00%	100.00%	100.00%
PART_11	100.00%	100.00%	100.00%
PART_12	100.00%	100.00%	100.00%
PART_13	84.00%	100.00%	76.47%
PART_14	100.00%	100.00%	100.00%
PART_15	100.00%	100.00%	100.00%
PART_16	100.00%	100.00%	100.00%
PART_17	100.00%	100.00%	100.00%
PART_18	100.00%	100.00%	100.00%
Confidence low	86.95%	90.81%	84.31%
AVERAGE	93.40%	95.90%	91.92%
Confidence up	99.84%	100.00%	99.54%

TABLE 8
Adequacy per Questionnaire

Questionnaire	Adequacy	Adequacy on Class Diagrams	Adequacy on Activity and Use Case Diagrams
PART_1	88.89%	90.00%	80.00%
PART_2	92.00%	90.91%	85.71%
PART_3	78.26%	100.00%	54.55%
PART_4	58.33%	37.50%	64.29%
PART_5	100.00%	100.00%	100.00%
PART_6	100.00%	100.00%	100.00%
PART_7	100.00%	100.00%	100.00%
PART_8	76.00%	87.50%	58.33%
PART_9	88.89%	100.00%	78.57%
PART_10	96.15%	100.00%	91.67%
PART_11	100.00%	100.00%	100.00%
PART_12	41.67%	50.00%	30.77%
PART_13	100.00%	100.00%	100.00%
PART_14	100.00%	100.00%	100.00%
PART_15	59.26%	72.73%	33.33%
PART_16	91.67%	100.00%	84.62%
PART_17	32.00%	33.33%	7.14%
PART_18	23.08%	10.00%	15.38%
Confidence low	66.57%	67.48%	55.77%
AVERAGE	79.23%	81.78%	71.35%
Confidence up	91.90%	96.08%	86.94%

of the "expert responders" that, thanks to the questions, can contribute to the validation of the input domain model.

Table 7 reports the data we obtained studying the efficacy of MOTHIA on the CHOReOS domain model. The average efficacy is associated with 95 percent confidence intervals, calculated using the Student's t-distribution. Overall, we see that for this case study MOTHIA scored a good result in terms of efficacy, with confidence intervals that span only ± 6 percent around the average values.

For the sake of completeness, Table 7 also reports a comparison between the efficacy obtained from the Class Diagrams only and the efficacy resulting from both the Activity and Use Case diagrams. In this case, the data revealed that the average efficacy on the Class Diagrams scored slightly better than the others.

From a deeper analysis of the collected data we identified that such a result is due to the fact that expert users were not sure what to answer (i.e., `DK`) more often to questions about Activity and Use Case diagrams (i.e., 22 times over a total of 30). This does not mean that people misclassified themselves as experts when they answered questions. Our justification is that during the definition of the CHOReOS domain model MEs and DEs spent more effort having common discussions on structural aspects (i.e., mainly modeled with the Class Diagrams) rather than on behavioral aspects (i.e., mainly modeled with the Activity Diagrams).

The *adequacy* of a generated questionnaire has been defined considering the number of questions classifying the interviewee as an "expert" of the specific topic addressed, over the total number of questions contained in the questionnaire. This metric is formulated in Equation (6):

$$\frac{\#Total - \#NE}{\#Total} \times 100. \tag{6}$$

Table 8 reports data about the adequacy of MOTHIA on the CHOReOS domain model. Even though some of the interviewees (e.g., questionnaires "PART_12", "PART_17", and "PART_18") marked quite a few answers with `NE`, most of the questions automatically generated using MOTHIA were in line with the expertise of the DEs. The adequacy for the rest of the questionnaires ranged between 58.33 and 100 percent, while the average score obtained considering the whole set of questionnaires is 79.23 percent, with a confidence range of ± 13 percent.

Although we cannot ignore the singularity reported by the three questionnaires referred above (which is also highlighted by the span of the confidence intervals), we cannot exclude that an extensive marking of the `NE` option can be ascribed to a lack of effort invested by some of the DEs in answering the questionnaires.

6 Correcting the Model from the Experts' Feedback

The CHOReOS domain model has been refined by considering the feedback provided by DEs when answering the questionnaires. It is important to mention that during one of the CHOReOS project meetings some weeks before distributing the questionnaires, the involved partners were informed about the experimentation, and explanatory material was shown and distributed. In particular, different kinds of diagrams were discussed in order to show the corresponding questions that MOTHIA is able to generate. Questionnaires were distributed by means of the Web-based `abQuestionnaire` engine that permits to stop and recover working sessions. In this way interviewees had the possibility to have a

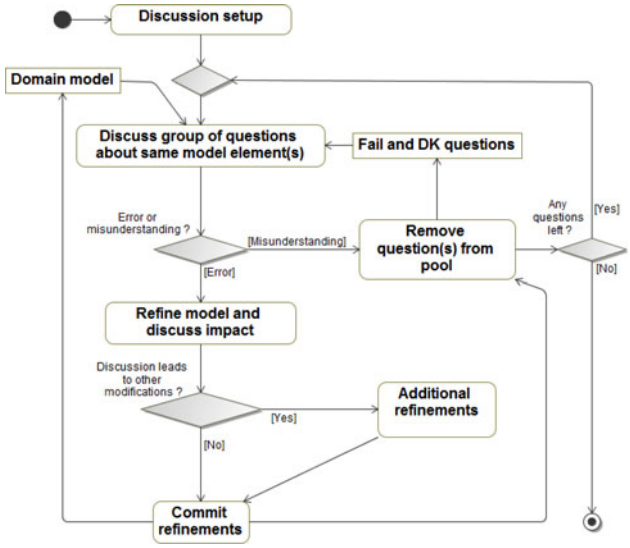


Fig. 7. Model refinement activities performed with the questionnaires interviewees.

look at the CHOReOS domain model or refer to additional material before answering questions. The refinement process performed after having collected all the answers, which is depicted in Fig. 7, corresponds to the sub-process *Domain Model Refinement* of the overall methodology shown in Fig. 1.

The refinement process focused on the answers that differed from the expected ones, more precisely on those categorized as `Fail` or `DK` in Table 5a. Both types of answers could hint at possible modeling errors: `Fail` refers to a DE conceiving some of the domain entities/relationships differently from the ones modeled by the MEs, whereas `DK` presumes some odd or unexpressed aspect of the model for an expert not to be able to express an opinion.

A phone call was set up with each DE who gave unexpected answers. This type of interaction between MEs and DEs, driven by the outcomes of the questionnaires, was necessary since we could not rely on the existence of an oracle as usually occurs in traditional testing processes. The discussion was focused on groups of questions, clustered according to the domain elements involved. The discussion started by asking feedback on the given answers, in order to identify whether the DE had misunderstood the question. If a real error was identified, its causes were investigated and discussed. The domain model was then refined accordingly. Interestingly, in some cases an error triggered modifications of elements that were not included in the original question (s). The discussion then continued with the next group of questions, until all the questions were analysed.

In line with the overall considerations on the validation of a system context given in [30], we identified four causes behind the most common errors. Specifically:

- *incorrect information*: the representation of some element was based on incorrect information about the domain; the MEs made wrong assumptions about the concepts presented by the DEs.
- *tacit assumptions*: most of the relevant elements of the domain model were modeled, but some aspects were not adequately considered; for example, the DEs did not mention an *evident* relationship among some domain elements, or a *major* property of one of them, taking them for granted.
- *semantic ambiguities*: the DEs underestimated the potential misunderstanding of the semantics of the terms used during the discussion with the MEs; for example, using either synonymous or improper terms that are similar but denote different domain elements.
- *unstated features*: one or multiple aspects of the domain model were overlooked since the early description of the model; for example, the DEs initially preferred to postpone their inclusion (e.g., to simplify the domain model), but then these aspects were left unstated.

In few cases, the discussion revealed that unexpected answers were not due to errors, rather to unclear formulation of questions. In such cases, DEs amended the previous given answer by recognizing that the model was correct and they gave the wrong answer. We provide in the following a qualitative analysis of errors detected in the model. We did not perform a detailed quantitative analysis. Just to give a measure of the impact that the questions among the genuine ones that were classified `Fail` had in the whole refinement process, 49 questions out of 94 generated ones (≈52 percent) triggered modifications of the model. We did not collect in this study precise metrics about effort spent on discussing unexpected answers not caused by a modeling error. However, their impact appeared limited. The discussion on such unclear `DK` answers was nevertheless useful to improve the formulation of NL questions; it is expected that as the tool matures, less and less `DK` answers should be collected.

In the remainder of the section we discuss some sample errors fitting the above classification. For each of them we also present the consequent refinements on the CHOReOS domain model.

6.1 Incorrect Information

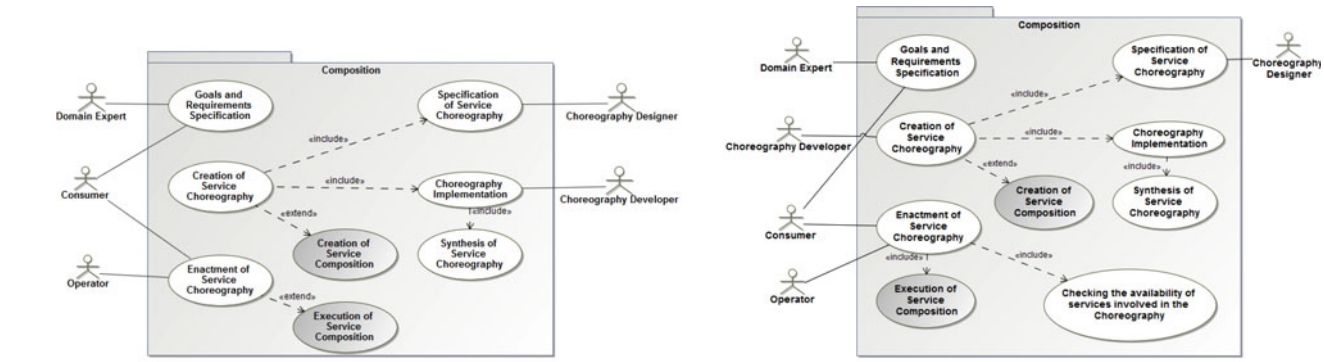
A first example of wrong assumption made by the MEs, which led to incorrect information in the domain model, is in the definition of the concepts about service compositions [13].

In the preliminary version of the domain model the ME kept the `Enactment of Service Choreography` as an independent behavior that could be used to augment the definition of the `Execution of Service Composition` use case (see Fig. 8a). Nevertheless, a positive answer to the Distractor:

> Is the use case `Execution of Service Composition` a part of the use case `Enactment of Service Choreography` ? (Q1)

revealed the DE was considering the enactment of a choreography as including the execution of a generic service composition. As a consequence, the domain model was refined and the relation connecting the two use cases was updated.

Another example about this category of errors lies in the definition of a continuous design process [13], and in particular, in the test activation functionality. Specifically, the model in Fig. 9a has been refined by discussing the Distractor:



(a) Initial model. (b) Refined model.

Fig. 8. Service compositions features.

> Is `Verification Event` always the previous action executed before the action `Activation Evaluation`? (Q2)

The answer to Question Q2 given by the DE was `Yes`, since the selection and the execution of a test can be done only after the verification of the event that triggered its activation. However, the expected answer computed by MOTHIA was `No`, because of an inaccuracy in the model. In particular, the initial node of the model has a direct link with the activity `Activity Evaluation`. According to the semantics presented in Section 3.2.3, which interprets multiple incoming flows as an *implicit merge*, such activity may become the first one to be performed, even before `Verification Event`. The ME represented this scenario as she unilaterally assumed that the activation of test functionalities without any specific event was also admissible. This required a modification of the model by changing the connection of the initial node, which has to point to `Verification Event` (see Fig. 9b).

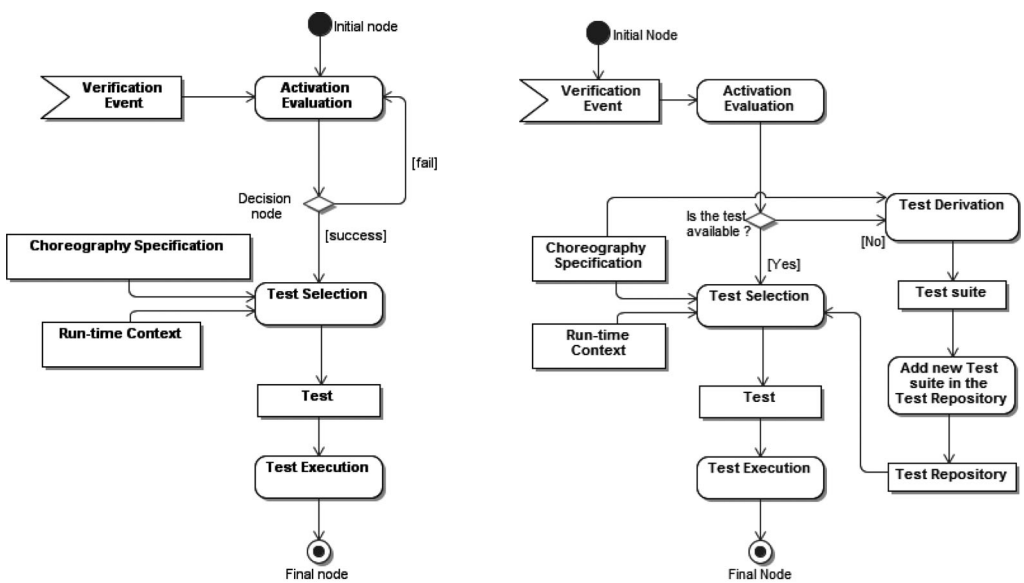
Note that the adoption of a single incoming and outgoing flow to/from an action is strongly recommended, showing all joins and merges explicitly [31]. In this sense, the refined model is even more robust to the possible semantics misunderstanding.

6.2 Tacit Assumptions

An example about the identification of a tacit assumption concerns the concepts that model the service compositions.

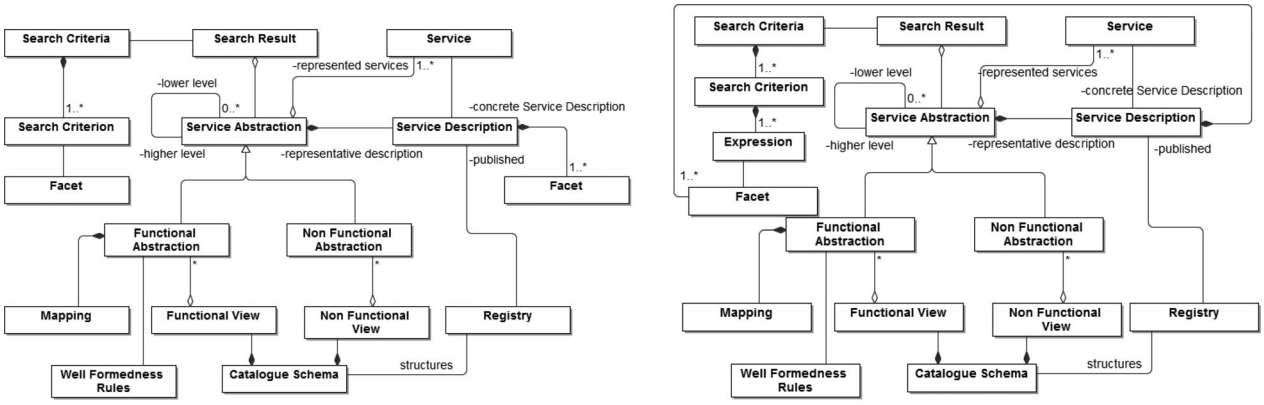
The initial version of the model is presented in Fig. 8a. During the resolution of the issue revealed by Question (Q1), the discussion with the DE led to realize that, while not modeled, the actor `Choreography Developer` could be involved in the use case `Creation of Service Choreography`. The DE asserted that this relation was for him intuitive and therefore not worth to be explicitly mentioned, while on their side the MEs did not spot this information. Fig. 8b depicts the refined version of the diagram.

For the sake of completeness it is important to remark that MOTHIA has been designed in order to address those errors that are mostly due to communication failures between DEs and MEs. In some cases, the errors that MOTHIA helps to discover can also be revealed by means of other approaches/techniques. In this specific scenario, as the use case `Creation of Service Choreography` in



(a) Initial model. (b) Refined model.

Fig. 9. Test Activation functionality.



(a) Initial model.

(b) Refined model.

Fig. 10. Structural entities supporting service discovery.

Fig. 8a does not have any relation with any actor, a static validation of the domain model would have been sufficient.

6.3 Semantic Ambiguities

Semantic ambiguities are the most common and difficult kind of errors that may happen in the communication between DEs and MEs. Considering the complexity of the problem [30], our work in MOTHIA does not pretend to provide a general or definitive solution to it. Nevertheless, in our case study we found that the discussion we foresee in the validation steps can contribute to highlight some of those problems.

As deeply investigated in the field of requirement engineering (RE) [30], these errors are often due to an awkward reference to different concepts by means of either terms that look similar, or synonymous. Specifically, let us refer to the diagrams modeling the functionalities that are needed to support the discovery of services [13]. Here, in the source model in Fig. 10a, an unexpected answer was received for the Distractor:

> Can a relation from `Service Description` to an unlimited number of `Facet` exist? (Q3)

After investigating the domain model, we realized that the MEs defined two entities, both named as `Facet` but located in two different packages, to express two different concepts. Note that in the model of Fig. 10a the two entities appear close in the same diagram only for the sake of presentation.

When generating Question Q3, MOTHIA considered the left-side element called `Facet`, thus the expected answer it computed was `No`. When answering the question, the DEs assumed that the element was the right-side one. Even more, during the following discussion it was clear that the two concepts were actually describing the same abstract entity. Consequently, they have been unified in the refined version of the domain model (see Fig. 10b).

For the sake of completeness, we remark that the predicates obtained through a domain ontology (i.e., Hypotheses) could improve the effectiveness of MOTHIA in dealing with semantic ambiguities. According to the examples given in Section 3.2, an ontology that organizes lexicon into a structured set of words, using relations among word meanings, could help to probe the real meaning of a word within the context of the domain model.

6.4 Unstated Features

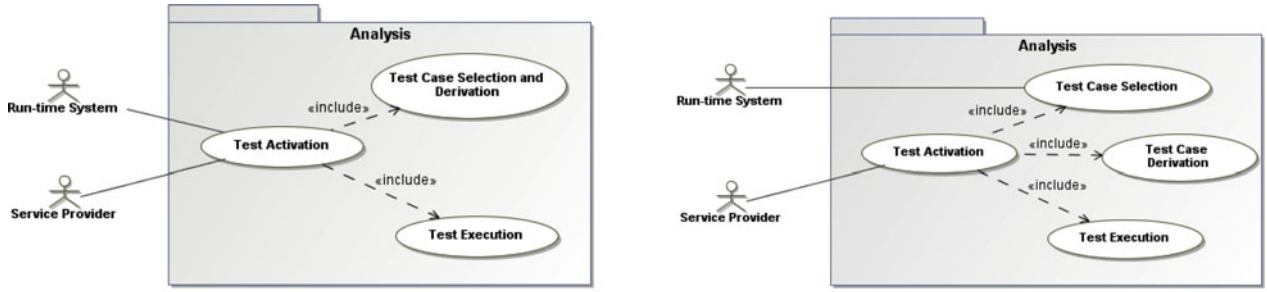
The last category of errors is mainly due to some preliminary simplification of the domain model that then led to unstated features. For most of these errors, there were no questions generated by MOTHIA that supported their direct identification. In fact, most of the concepts fitting in this category were not included in the model at all. Nevertheless, the combination of the methodology presented in this paper and the systematic generation of questionnaires by MOTHIA, triggered a thorough discussion about the produced artifacts. Through that, a few errors related to unstated features could be revealed.

As an example, while tackling the incorrect information included in the model in Fig. 8a (see Section 6.1), the discussion led to realize that the enactment of a service choreography (as modeled) may implicitly assume that all the services are already available. This is not true in the general case, thus DEs preferred to explicitly model a separated use case dedicated to the verification of the availability of all the required services as part of the choreography enactment (see Fig. 8b).

Similarly, the discussion with the DE about the incorrect information revealed by Question Q2 pointed out some other issues in the model in Fig. 9a. In fact, the DE completely omitted the specification of a decision branch about the derivation of new test suites, which can be required to test a given choreography specification. Thus, the model has been corrected as in Fig. 9b, by adding the new activities `Test Derivation` and `Add new Test suite in the Test Repository`, and connecting them to the rest of the diagram. The concept is that, given a choreography specification to be tested, a first check is performed in order to see if the relative test suite is available in test repository. If it is not available, it is derived instead and added to the repository. The new test suite can then be executed.

Interestingly, the discussion on the models in Fig. 9 also led to refine the model in Fig. 11, in order to explicitly represent the `Test Case Derivation` functionality. The actor `Run-time System` has been modified by removing the association with the `Test Activation` use case (which is performed only by the `Service Provider`), and by adding the relation with the use case `Test Case Selection`.

As discussed in Section 6.3, we foresee that also for this class of errors the use of Hypotheses should mitigate the risk



(a) Initial model. (b) Refined model.

Fig. 11. The Use Cases enabling the testing activities.

of missing unstated concepts in the domain model, as they leverage on psycholinguistic theories of human lexical memory [16]. Specifically, the adoption of relationships between concepts (e.g., hyponym relations) are used to expand and disambiguate clustered concepts, in order to increase the likelihood of an accurate domain model exploration.

7 Comparison with the Previous Experience

The results discussed so far have to be interpreted in the context of the CHOReOS case study and are not supposed to be used for formulating generally valid considerations. Nevertheless, as our goal is to get some empirical validation of our approach, repeating the validation on more case studies becomes necessary [12]. In the already cited previous paper [11] we reported about a similar yet independent case study conducted in the context of the IPERMOB project. An *ex-post* comparison of the results obtained for IPERMOB and CHOReOS can be interesting to evaluate both the methodology and the framework we developed. Furthermore, as in this paper we extend the original framework, such a comparison provides also a way to evaluate if/how such extensions improve the effectiveness and applicability of the approach.

7.1 Dimensions

We identified eight different dimensions along which to compare the two experiences, and highlight differences and similarities.

D1—Size: The IPERMOB domain model consisted of 50 conceptual entities (modeled as classes) grouped in six different packages. The overall amount of relationships modeling the semantics connection (i.e., associations) among such conceptual entities was 68, 14 of which represented special kind of semantic interconnections that relate to structural aspects of the entities (i.e., compositions). Finally, the domain model included 23 relations denoting a taxonomic relationship between a more general entity and some specific ones (i.e., generalizations). All such aspects concerned only the structural specification of the domain model.

As already introduced in Section 4.1, the domain model in CHOReOS was roughly double in size when compared with IPERMOB. Furthermore, the new features added to MOTHIA supported a wider validation of how MEs modeled the behavioral aspects intended by the DEs.

D2—Number of questionnaires: In the two case studies, the number of projects partners available for the validation of the domain models were close. Specifically, with respect to IPERMOB we were able to enroll 14 interviewees, while in CHOReOS we counted on 18.

In our view, this dimension defines our capability in validating both the input domain model and the framework we are proposing (with respect to the faulty questions). In relative terms, based on the considerations given in the dimension D7.1, we can assert that the CHOReOS case study addressed a wider problem with almost the same resources as the IPERMOB one.

D3—Distribution of questionnaires: The IPERMOB project was composed of partners from a limited geographical area (i.e., a 100 Km radius). This context made it easy to set up in-person meetings with DEs, where the submission of questionnaires was aided by examples. Nevertheless, DEs were supposed to answer each question on the basis of their knowledge, without consulting any kind of associated documentation.

On the contrary, CHOReOS is a European project, with partners located all over Europe, and in Brazil. The availability of a web-engine such as abQuestionnaire was a crucial asset in making MOTHIA effective in such a distributed scenario. In addition, due to the increased complexity of the CHOReOS domain model, it was not realistic to assume that the DEs were able to promptly remember all its details. Thus, we introduced a framework to store partial sessions over time, giving the possibility to DEs to pause their compilation, consult any project documentation, reflect on the context of the question, and then resume the answering of the questionnaire. In our opinion, this feature contributed to the good results that MOTHIA showed in this case study.

Finally, another important distinction between the two distribution processes is that in IPERMOB all the partners were committed to take part in the case study by project agreements, whereas in CHOReOS the partners took part in the case study on a voluntary basis. Thus, we did not have full control over the skills that were actually dedicated to the case study.

D4—Fault types: During the definition of the case study on IPERMOB, we developed an engine to automatically create mutants of the input domain model by randomly inserting faults in it. Specifically, such engine was able to deal with five types of faults: four of them were altering the semantic connection among the domain entities (i.e., the associations), while one was dealing with the generalization relation.

With respect to the mutation of the associations, the engine was able to create new relationships, invert the direction of an existing association, change its cardinality, or

specialize a regular association into a composition and vice versa. With respect to the generalization, the engine supported either the creation of a new generalization connection, or the replacement of an element with a parent or sibling.

In this work, we improved the mutation engine to support abstract faults, applicable on a graph-based abstraction of the diagrams. Indeed, as reported in Table 4, in CHOReOS we used five types of abstract faults: specifically, the creation, the modification (in terms of endpoint, direction and type), and the deletion of any relationship represented by an edge.

In this way, the mutations are agnostic with respect to the considered elements. Thus, their application can span over the whole domain model. This contributes towards the mitigation of the bias in the validation of the methodology (see Section 5).

D5—Number of faults: Considering the types of faults described above, in IPERMOB we introduced five mutations into the input domain model (one per fault type). For each type of fault at least one question was generated covering the mutation. The number of faulty questions generated by those faults is described in the next dimension.

In this case study the mutation engine was configured to introduce 22 mutations in the model. Such a number of mutations was obtained by planning a single alteration for each diagram contained in the CHOReOS domain model, in order to evenly distribute them across the domain. This decision was made because, unlike the IPERMOB model (a purely structural diagram where every entity could potentially be interconnected with each other), the CHOReOS model includes various independent behavioral diagrams, so that a fault is isolated within its own diagram. Both the type of fault and the model element designed for the modification were randomly drawn, but we ensured that every type of fault was applied at least once.

Nevertheless, after the execution of MOTHIA we noticed that only 19 of those mutations were actually reflected in the generated questionnaires. In other words, for three instantiations of the mutations there was no faulty question generated from the mutated domain model. This can be justified by means of the Faults Density metrics in the dimension D8.

D6—Faulty questions: In both case studies we tried to keep a fixed rate of errors per questionnaire (approximately five questions per questionnaire). Such procedure was achieved by fine tuning the filtering rules of MOTHIA, following the actual generation process (see Section 3.1). In IPERMOB the total number of faulty questions was 78, while in this case study was 88.

The 88 faulty questions were originated due to the mutations discussed by dimension D5; in other words, each mutation affected one or more faulty questions.

D7—Faults found: According to the data presented in [11], we can summarize that, with regards to the artificial faults in IPERMOB, the answers to the questionnaires were able to: a) detect all types of faults (i.e., five types of faults); b) detect all the faults that mutated the domain model (i.e., 5/5 faults); c) kill around half of the faulty questions (i.e., 58 percent).

In this case study: a) all the types of faults were detected (i.e., five types of faults); b) most of the mutations were revealed (i.e., 16/19 faults, 84.21 percent); c) the 60.32 percent of faulty questions were killed. For the sake of clarity we remark that, according to dimension D5, after the execution of MOTHIA three of the mutations did not appear in any of the generated questionnaires. Thus, the total amount of discoverable mutations were 19 rather than 22.

In this sense, considering the increase in complexity of the CHOReOS case study (see dimensions `Size`, and `Number of Questionnaires`), we can assert that the improvements added to MOTHIA have led to comparable good results to those in IPERMOB, even though the framework was applied with similar resources on a wider (and more complex) domain model.

D8—Faults density: We elaborated a final dimension to contribute in interpreting positively the results discussed in the previous dimensions. Specifically, we introduced two new metrics:

$$\text{FaultyQuestionsDensity} : \frac{\#FaultyQuestions}{\#Questions} \times 100, \quad (7)$$

$$\text{FaultImpact} : \frac{\#FaultyQuestions}{\#Faults}. \quad (8)$$

The Faulty Questions Density (i.e., Equation (7)) denotes the percentage of faulty questions in the generated questionnaires. According to the data reported in the previous dimensions, in IPERMOB we had 18.57 percent (i.e., 78/420), while in CHOReOS 16.30 percent (i.e., 88/540).

The Fault Impact (i.e., Equation (8)) estimates the number of faulty questions dedicated to each fault that mutated the domain model. The greater this factor, the higher the chances to kill the mutation each faulty question subsumes. With respect to this metric, IPERMOB has 15.6 (i.e., 78/5) faulty questions per fault, and CHOReOS has 4.63 (i.e., 88/19) faulty questions per fault.

Thus, from a comparable overall potential for fault detection (i.e., Equation (7)), the scenario run within CHOReOS was more difficult than the one run within IPERMOB with respect to the detectability of each single fault (i.e., Equation (8)).

7.2 Lessons Learned

From the application of MOTHIA to two similar but different contexts, as well as from the observation of the additional features compared to [11], we learned several interesting lessons.

First of all, supporting the validation of behavioral diagrams helped DEs and MEs in getting a better knowledge of the modeled domain entities. Even though we plan to further extend MOTHIA to support other kinds of notations and cross relationships between them, we could probably investigate first other sets of taxonomies for both patterns and criteria, especially when considering behavioral diagrams.

Also, we clearly confirm the fact that the interaction with human subjects requires a strong motivation and commitment for all the involved people. In both our experiences we somehow noticed that DEs perceived these validation steps as an additional wearing task; thus, they usually tend to postpone it as much as possible. The lesson learned: it is

important to let the DEs understand how fundamental is their cooperation, but also activating the validation process as soon as an iteration of the modeling phase ends.

The adoption of a web-based distribution engine for the questionnaires was perceived in a quite positive way. Offline, aside from the experiment, DEs reported that it gave them time to actually understand the questions, and to consult other documentation (e.g., emails exchanged) about the entities related to the questions; on the contrary in IPER-MOB, with face-to-face sessions, they had the perception to be examined. From our perspective, it is also important to agree with the DEs on some kind of remote assistance service, due to the possible misunderstandings (e.g., naturalness of the questions in Section 8).

The results from each experiment, as well as the comparison with results from different experiences, can be used to iteratively fine-tune the many configurations allowed by MOTHIA. The filtering of the generated questions is an area that could particularly benefit from such approach. For example, the impact of each model element on the number of `Fail` questions, or its coverage on the total number of questions are interesting aspects to potentially investigate. Similarly, the same procedure could be also applied to the impact and coverage of each criterion.

This paper has mainly focused on both efficacy and adequacy metrics, as defined in Section 5.2. Clearly, additional metrics such as the ones hinted above could be adopted to judge the quality of the questionnaires, and to further assess MOTHIA. A triangulation of metrics would support the evolution of the approach from the current version that adopts a random-based selection, to a version that adopts weighted criteria, based on the impact in discovering errors and the coverage of elements in the model.

8 Threats to Validity

In this section we discuss the threats to validity that could affect our results by distinguishing threats to *construct*, *external*, and *internal* validity.

Threats to construct validity concern the appropriateness of our measures for capturing our dependent variables. In our study, we targeted two different objectives: validating the CHOReOS domain model, and evaluating MOTHIA. To this end, we generated questionnaires consisting of 25 questions for the validation of the CHOReOS domain model, and five questions for evaluating the percentage of faults revealed by MOTHIA (given that a fault is covered by a questionnaire). In this respect, we identify the following threats to construct validity:

Number of questions: the number of questions (i.e., 30) composing each questionnaire was decided in order to have a questionnaire requiring less than one hour to be filled. A longer questionnaire could compromise the quality of the answers due to the reduction of attention and concentration of the people involved in the case study.

Proportion of the two different kinds of questions: because of the dual nature of our study, changing the proportion of the two kinds of questions would affect the focus of the experiment. Since our main goal was to validate the CHOReOS domain model, we decided to produce more questions for that purpose (450 of 540 questions) and to limit the number of questions related to the validation of MOTHIA to 90.

Evaluation metrics: as discussed in Section 5, the evaluation of the results collected in the presented case study relies on the *efficacy* and *adequacy* metrics. This choice impacts the conclusions that can be drawn from our study. Among the lessons learned presented in Section 7, we argued that further metrics could be adopted. Nevertheless, we decided to focus on efficacy and adequacy since they permitted to compare the results of the experience presented in this paper with that discussed in [11].

Threats to external validity refer to the extent to which the results of our study can be generalized. In this respect, we identify the following threats to external validity:

Number of DEs: the performed experience is affected by the number of DEs who have been interviewed and the number of questions composing each questionnaire. In this respect, the collected data can have a reduced validity from a statistical point of view. However, as discussed in the paper, the experience has been done in the context of a large European project, where partners involvement in activities that are marginal to the central objectives of the project has been difficult. This motivates why we tried to define a trade-off in terms of number of people involved in the case study, their expertise in the field, and number of questions.

MEs background: MEs belonged to those partners of the consortium that explicitly introduced modeling phases within the CHOReOS 's activities. As MEs were member of the project as well, it is undeniable that they had some previous knowledge about the target domain model.

Naturalness of the questions: sometimes the NL questions produced by the approach can appear not realistic, i.e., it is evident that they have been generated by an automated process. This aspect might hamper the comprehension of the questions and thus the validity of the given answers. For instance, the question shown in Fig. 6 *"Can the relation 'Notifies' from Event Broker to V&V Manager exist?"* might be better rephrased as *"Can the Event Broker notify the V&V Manager?"* in order to have it closer to the application domain. To deal with such a problem, the approach can be extended by borrowing concepts and techniques coming from the area of Computational Linguistics. For instance, there are methods to fingerprint the structure of English sentences that might be adopted to extend the question generator of MOTHIA in order to generate sentences closer to natural language.

Medium for questions: DEs were interviewed by means of Web-based questionnaires. The alternative would have been running personal interviews. As discussed among the lessons learned in Section 7.2, having used the web as the medium for performing the case study allowed DEs to take their time to answer questions, to view the documentation about the model fragments involved in the questionnaires, and think about them without any time pressure. Once all the answers were collected and analysed, we contacted DEs by phone or by email to discuss the answers that were different from the expected ones. We recognise that the access to supplementary documentation could have threatened the performance of MOTHIA to reveal errors. Nevertheless, we also consider useful the fact the DEs were stimulated in revising any additional elaboration about the domain model.

Type of projects: the case study we have discussed in the paper has been done in the context of CHOReOS i.e., a large EU project consisting of 15 partners having different expertise and involvement in the project. Performing similar case studies in projects that are different from CHOReOS in terms of size or partners inclination towards modeling activities might produce different results. To deal with such a problem, before distributing questionnaires it is important to instruct partners by showing representative diagrams and discussing the corresponding questions generated by MOTHIA. In this respect, during one of the CHOReOS meetings we informed the partners that they would have been involved in the case study we were setting up, we showed a number of sample diagrams of the CHOReOS conceptual model, and discussed the corresponding questions generated by MOTHIA.

Choice of patterns and criteria: the generation of questions in MOTHIA relies on a set of patterns and criteria that were selected according to the experience of the authors. Considering other patterns and criteria would imply the generation of different questionnaires. For this reason, the patterns and criteria used are generic, not domain specific, and based on the structure of the diagrams rather than their semantics. As said in Section 4.2, in this paper we proposed a general methodology and related supporting tools, without pretending to generalize the results to any set of patterns/criteria.

Filtering of questions: in the reported case study we have filtered questions to ensure that a certain number of faulty questions was included in each questionnaire. As the number of questions that can be generated by MOTHIA are infeasible to answer by DEs within a reasonable time, in real-life situations filtering is necessary to select questions that are likely to address potential issues in the examined model. Currently we are not providing indications about any filtering policy, this is a limitation of the approach and future work will need to explore effective filtering techniques.

Threats to internal validity refer to the extent to which the results obtained are function of the variables that have been systematically manipulated, measured, and observed in the study. In this respect, we identify the following threats to internal validity:

Kinds of faults mutating the diagrams: the faults introduced in the case study were related to structural defects in the model, rather than its semantics. In other words, we did not consider the meaning of the involved model elements when applying the mutations. This is in line with the principle of mutation in testing.

Diagram mutations: we randomly mutated each diagram of the CHOReOS domain model on a single element in order to generate faulty questions. As we described in Section 4.2, our strategy was to generate questionnaires characterized by a fixed rate of injected errors (i.e., five faulty questions per questionnaire). This might have created a bias, which we tried to limit by uniformly distributing the faults, and by formulating general principles according to the know-how gained in the IPERMOB project. Also, we did not use the obtained results to infer a generic percentage of faults detection capability.

Completeness of the results: one last threat to validity is related to the completeness of the results; in this sense, we are not able to assess neither that we have discovered all the problems, nor how many and what faults remain undiscovered. However, this is a typical problem affecting any approach that, like the one presented in the paper, is not exhaustive.

9 Related Work

Other works exist that use questionnaires for model validation. The work in [32] proposes questionnaire-based models that, including order dependencies and domain constraints, allows for customizing configurable business processes. Analysis techniques are proposed for detecting circular dependencies and contradictory constraints in the questionnaire models and for preventing invalid configurations. For each question, the proposed techniques restrict the space of allowed answers based on previous answers.

In [33], the authors propose a technique to generate multiple-choice tests starting from electronic questions. The approach differs from ours in several aspects: their goal is to produce tests that are useful to test the expertise of the responders in the area covered by the questionnaire; they use NL analysis to extract knowledge from a corpus and to formulate questions related to that corpus. Our goal is to test the model, not the responder's knowledge.

The approach proposed in [34] uses machine learning techniques to generate questions of fill-in-the-blank type, in order to allow the testing of any kind of knowledge, rather then for a specific purpose like our approach.

The automatic creation of NL multiple-choice questions from domain ontologies is proposed in [19]. The goal is to create a questionnaire for educational purposes, rather than validation, since the input knowledge base is considered as correct. While we focused on simpler Yes/No questions, strategies to create distractors can be seen as the equivalent of our criteria.

Considering goal models and business process models as complementary artifacts when capturing the requirements and their execution flow, in [35], the authors make use of Description Logics (DL) [36] and automated reasoners to validate mappings between goals and their realization in term of activities in business process models. The approach uses DL to model workflow patterns and automatically check if executable processes do not meet user intentions and needs, or lead to undesired executions. A set of predefined realization inconsistencies is used.

Differently from us, most of the above mentioned approaches only consider the structural part of system modelling, except for the approaches in [32], [35] that consider only business process models. In line with software engineering best practices, which tend to distinguish system descriptions into structure, behaviour, and functions [30], our approach can validate multi-views system models, being able to handle Use Case and Activity Diagrams, in addition to Class Diagrams.

If we consider a broader perspective about the definition of domain models, the literature comes with many evidences of how some kinds of properties/constraints require other mechanisms than only UML Diagrams to be properly expressed. For example, domain engineers usually express constraints in OCL because multiplicities are not enough to represent them in the UML Class Diagrams. In this sense,

domain engineers that aims at validating such constraints have to rely on some kind of complementary means.

Several approaches attempt at formalising (part of) UML for the purpose of (possibly) automatic reasoning. Considering a decomposition of the quality of conceptual schemas at syntactic level, semantic level, and pragmatic level (as in [6]), the work in [37] focuses on early assessment of semantic qualities of UML conceptual schemas, with OCL integrity constraints. Two different perspectives are considered: from an internal point of view verification techniques are applied to check whether the schema contains contradictions and redundancies; from an external point of view validation techniques are applied to check whether the schema fulfills the requirements of the application it is being built for, while properly representing the knowledge about the application domain. Considering the undecidability of the problem of automatically reasoning with arbitrary OCL constraints, the proposed approach is a compromise between dealing with arbitrary constraints, for which decidability is not guaranteed, and checking whether termination is ensured or not. To assess correctness while ensuring termination, a first-order logic formalization is used to encode the structural part of a UML schema together with its OCL constraints. Then, according to the logic representation, a set of questions is formalized as derived predicates and, as such, can be answered by checking satisfiability of predicates. A predefined set of "internal" and "external" questions can be automatically verified and validated (respectively) on any schema, and are an original contribution of the work in [37]. Other properties that cannot be covered by the predefined predicates must be extracted by the designer from the requirements of each particular application, and must be formalized.

The work in [38] focusses on the semantic quality validation of model transformations when used to refine a source model into a target model. The proposed approach checks the correctness of a target MOF/UML models with regards to the corresponding source model by using OCL to encode refinement simulation conditions. These conditions are then automatically evaluated by combining model checking, testing and semantic entailment.

Recognizing the importance of analyzing system models at early stages of the software development lifecycle (to mitigate the risk of additional cost and effort when implementing a system based on a faulty design), the work in [39] proposes an approach to validate a UML/OCL schema by encoding a subset of UML and OCL into the Alloy logic formalization [40]. Alloy[7] is a structural modelling language based on first-order logic, for expressing complex structural constraints and behaviour. Considering only UML Class Diagrams, the analyser proposed in [39] implements a "solver" that, given a logical formula in Alloy language, attempts to find a model (i.e., a binding of the variables to values) that makes the formula true. In other words, it takes as input properties specified by the designer, and it searches for examples matching them. The analyzer automatically explores the state space exhaustively, up to the user specified scope. The approach allows for general OCL constraints (with some restrictions) without guaranteeing completeness of the result and does not support association classes and *n*-ary associations.

In the previous approaches, the usage of logic formalisms, formal analysis and checking tools may be error prone and often requires high expertise. For example, as admitted by the authors in [37], a high expertise may be required (*i*) to correctly formulate the right questions by using the proposed approach, and (*ii*) to check their satisfiability by using satisfiability checking methods. This consideration reinforces our thesis that DEs may not be able to directly check the models and MEs are called to interpret the needs of DEs, translate them into formal notations, and finally try to understand if what is formalized (and then checked) correctly represents what was in the mind of the problem owner. Note that, although MOTHIA works on a first-order logic representation of the input model, the end-user is not required to understand it.

In the context of this work, it is also worth mentioning requirement engineering (RE) approaches to elicit knowledge from experts and validate requirements descriptions, which are well recognized and widely accepted in the literature.

In [41], [42] (and references therein) the authors provide a research perspective and a roadmap that clarify how the RE discipline covers multiple intertwined activities, namely: (*i*) domain analysis to identify relevant stakeholders to be interviewed, hence studying the context and general objectives in which the software should be built; (*ii*) elicitation to explore alternative models for the target system, and define requirements and assumptions of such models, hence meeting the identified objectives; (*iii*) negotiation and agreement to evaluate alternative requirements/assumptions; (*iv*) specification to precisely formulate requirements and assumptions; (*v*) specification analysis to check the specifications for, e.g., inadequacy, incompleteness or inconsistency, and feasibility; (*vi*) documentation to document the decisions made, and the underlying rationale and assumptions; (*vii*) evolution to account for requirements modifications, environmental changes, or new objectives.

Seminal work on elicitation techniques [43] spans from structured and unstructured interviews [44], [45], protocol analysis [46], card sorting [47], laddering [48].

As already clarified, our approach focuses on validating the very first domain models, rather than eliciting knowledge from experts and validating requirements descriptions. Specifically, our assumption is that such domain models are manually created according to requirements already elicited from the interaction with the owners of the problem domain. Here we refer to those domain models that are to be taken as input by `step zero` MDE model refinement techniques. Thus, for a proper and successful adoption of such techniques, models need to be correct not only syntactically but also semantically, according to what the domain experts have in mind. In the perspective of [41], [42], it can be said that our approach can be collocated within the specification analysis activity. That is, it can be considered as a means for checking the quality of early domain models that, being derived from requirements negotiated and agreed with DEs, show enough technical details and adequate technical precision to be amenable for model refinements through automatic MDE techniques.

7. http://alloy.mit.edu

10 Conclusions and Future Work

The definition of models and abstractions, as well as the domain-specific customizations of the modeling environment, are effective approaches that aim at drawing the developers closer to the problem domain. Nevertheless, research in MDE recognized that these are often error-prone activities that require synergies and skills going beyond the mere technologies used [2], [49]. Specifically, if on the one hand the MEs are familiar with technological notations, environments and resources, on the other hand they usually lack a deep knowledge of the application domain that requires to be analyzed and engineered.

The approach we promote in this paper contributes in supporting MEs in the early stages of an MDE process. We investigate how to test the validity of the models built at `step zero` by facilitating the interaction between DEs and MEs since the initial definition of a domain model. This goal is achieved by a systematic generation of questionnaires (Yes/No questions expressed in NL) that MEs can submit to DEs on the basis of the domain models currently elaborated. As for any testing approach, the contribution of MOTHIA is subject to the likelihood of selecting a question covering a model issue among all the possible questions that could be generated. In its current version, the support to such a selection is limited.

We obtained positive feedbacks concerning the effectiveness of MOTHIA in revealing modeling errors, both from the reported study in estimating how likely artificial faults exposed in a questionnaire are identified by a responding ME, and also from the qualitative analysis of interaction with DEs who gave unexpected answers. In addition, we also saw how MOTHIA can contribute to reducing the knowledge gap between DEs and MEs, which is often a source for early stage modeling errors. We believe that the new features introduced in this paper with respect to our previous work [11] improved the applicability and expressiveness of the framework, covering a bigger set of source domain models.

Future research will devote further efforts in automating the process illustrated in Fig. 1. We intend to support the analysis of the DEs responses to map the *discovered wrong answers* on the model elements they refer to. We will also explore more systematic ways to transform and correct the input models, which we currently handle manually.

In addition, the effectiveness of MOTHIA in detecting faults is related to the formulation of both patterns and criteria. The performance of the validation strategy can be impacted if criteria built around certain syntactical structures (i.e., patterns), rather than others, lead more often to questions that allow for detecting errors. Thus, work is also planned to study the characteristics and the taxonomies of both patterns and criteria. As an example, we plan to add scheduling analysis capabilities around the *graphFlow* pattern in the Activity Diagrams, to properly address concurrency problems.

As presented in Section 3.1, MOTHIA is also able to generate questions about elements that do not exist in the model. To this end, Hypotheses are generated by exploiting semantic relations among sets of cognitive synonyms (i.e., synsets) [50] that include model elements. As explained, this feature could not be validated in the context of the CHOReOS project. A stimulating direction we will undertake concerns the design of a specific experience where MOTHIA can be used to investigate the completeness of the considered domain model.

As mentioned in Section 3.2, MOTHIA does not handle cross-references among different diagrams yet. In some sense, it assumes that all diagrams are at the same level of abstraction, which is not usually the case. For example, Activity Diagrams can be adopted to describe Use Cases, or more fine-grained activities. This information could be used to better shape the questions on each specific context. Future work may also investigate how to improve such aspects.

Finally, we intend to focus future work in getting higher confidence on the results obtained by this and the previous experimentation [11], according to the guidelines given in [12]. In particular, we plan to repeat the empirical evaluation of MOTHIA on other models, possibly with more interviewees, and with an expanded set of artificial faults.

Acknowledgments

This paper describes work undertaken in the context of the European Project FP7 IP 257178: CHOReOS. The authors would like to thank Alessandro Baroni for his important contribution to the implementation of the web-based engine supporting MOTHIA.

References

[1] J. Bézivin, "On the unification power of models," *Softw. Syst. Model.*, vol. 4, no. 2, pp. 171–188, 2005.

[2] R. B. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *Proc. Future Softw. Eng.*, 2007, pp. 37–54.

[3] D. Di Ruscio, L. Iovino, and A. Pierantonio, "Coupled evolution in model-driven engineering," *IEEE Softw.*, vol. 29, no. 6, pp. 78–84, Nov./Dec. 2012.

[4] R. Paige and D. Varrò, "Lessons learned from building model-driven development tools," *Softw. Syst. Model.*, vol. 11, pp. 527–539, 2012.

[5] H. Nelson, G. Poels, M. Genero, and M. Piattini, "A conceptual modeling quality framework," *Softw. Quality J.*, vol. 20, pp. 201–228, 2012.

[6] O. I. Lindland, G. Sindre, and A. Sølvberg, "Understanding quality in conceptual modeling," *IEEE Softw.*, vol. 11, no. 2, pp. 42–49, Mar. 1994.

[7] J. Krogstie and A. Sølvberg. (2000). *Information Systems Engineering: Conceptual Modeling in a Quality Perspective*. The Norwegian Univ. Science Technol. [Online]. Available: http://www.idi.ntnu.no/ sif8060/03/bok.pdf

[8] R. Salay, M. Famelis, and M. Chechik, "Language independent refinement using partial modeling," in *Proc. 15th Int. Conf. Fundam. Approaches Softw. Eng.*, 2012, pp. 224–239.

[9] M. Autili, V. Cortellessa, D. Di Ruscio, P. Inverardi, P. Pelliccione, and M. Tivoli, "Integration architecture synthesis for taming uncertainty in the digital space," in *Proc. 17th Monterey Workshop Large-Scale Complex IT Systems. Develop., Operation Manag.*, 2012, pp. 118–131.

[10] R. Salay, J. Gorzny, and M. Chechik, "Change propagation due to uncertainty change," in *Proc. 16th Int. Conf. Fundam. Approaches Softw. Eng.*, 2013, pp. 21–36.

[11] A. Bertolino, G. De Angelis, A. Di Sandro, and A. Sabetta, "Is my model right? let me ask the expert," *J. Syst. Softw.*, vol. 84, no. 7, pp. 1089–1099, 2011.

[12] W. Tichy, "Hints for reviewing empirical work in software engineering," *Empirical Softw. Eng.*, vol. 5, no. 4, pp. 309–312, 2000.

[13] A. Ben Hamida, F. Kon, G. Ansaldi Oliva, C. Moreira Dos Santos, J. Lorré, M. Autili, G. De Angelis, A. Zarras, N. Georgantas, V. Issarny, and A. Bertolino, "An integrated development and runtime environment for the future internet," in *The Future Internet - Future Internet Assembly 2012: From Promises to Reality*, F. Alvarez et al., Eds., vol. 7281. New York, NY, USA: Springer, 2012, pp. 81–92.

[14] M. Broy and M. Cengarle, "UML formal semantics: Lessons learned," *Softw. Syst. Model.*, vol. 10, no. 4, pp. 441–446, 2011.
[15] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. Grose, *Eclipse Modeling Framework*. Reading, MA, USA: Addison-Wesley, 2003.
[16] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller, "Introduction to WordNet: An on-line lexical database*," *Int. J. Lexicography*, vol. 3, no. 4, pp. 235–244, Dec. 1990.
[17] M. Pezzè and M. Young, *Software Testing and Analysis - Process, Principles and Techniques*. New York, NY, USA: Wiley, 2007.
[18] H. Dalianis, "A method for validating a conceptual model by natural language discourse generation," in *Proc. 4th Int. Conf. Adv. Inf. Syst. Eng.*, 1992, pp. 425–444.
[19] A. Papasalouros, K. Kanaris, and K. Kotis, "Automatic generation of multiple choice questions from domain ontologies," in *Proc. e-Learning*, 2008, pp. 427–434.
[20] A. Cicchetti, D. Di Ruscio, L. Iovino, and A. Pierantonio, "Managing the evolution of data-intensive web applications by model-driven techniques," *Softw. Syst. Model.*, vol. 12, no. 1, pp. 53–83, Feb. 2013.
[21] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in property specifications for finite-state verification," in *Proc. 21st Int. Conf. Softw. Eng.*, 1999, pp. 411–420.
[22] S. Konrad and B. H. C. Cheng, "Real-time specification patterns," in *Proc. Int. Conf. Softw. Eng.*, 2005, pp. 372–381.
[23] C. Peltz, "Web services orchestration and choreography," *IEEE Comput.*, vol. 36, no. 10, pp. 46–52, Oct. 2003.
[24] N. Wainwright and N. Papanikolaou, "Introduction: The FIA research roadmap, priorities for future internet research," in *The Future Internet*, series Lecture Notes in Computer Science, F. Álvarez, F. Cleary, P. Daras, J. Domingue, A. Galis, A. Garcia, A. Gavras, S. Karnourskos, S. Krco, M.-S. Li, V. Lotz, H. Müller, E. Salvadori, A.-M. Sassen, H. Schaffers, B. Stiller, G. Tselentis, P. Turkama, and T. Zahariadis, Eds., Heidelberg, Germany: Springer, 2012, vol. 7281, pp. 1–5.
[25] V. Issarny, N. Georgantas, S. Hachem, A. Zarras, P. Vassiliadis, M. Autili, M. A. Gerosa, and A. Ben Hamida, "Service-oriented middleware for the future internet: State of the art and research directions," *J. Internet Services Appl*, vol. 2, no. 1, pp. 23–45, 2011.
[26] G. De Angelis, A. Bertolino, and A. Polini, "Validation and verification policies for governance of service choreographies," in *Proc. 8th Int. Conf. Web Inf. Syst. Technol.*, Apr. 2012, pp. 86–102.
[27] A. Bertolino, G. De Angelis, and A. Polini, "Governance policies for verification and validation of service choreographies," in *Proc. 8th Int. Conf. Web Inf. Syst. Technol. (Selected Papers)*, 2013, pp. 86–102.
[28] J. Mottu, B. Baudry, and Y. Le Traon, "Mutation analysis testing for model transformations," in *Proc. 2nd Eur. Conf. Model Driven Archit.: Found. Appl.*, 2006, pp. 376–390.
[29] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Trans. Softw. Eng.*, vol. 37, no. 5, pp. 649–678, Sep./Oct. 2011.
[30] K. Pohl, *Requirements Engineering - Fundamentals, Principles, and Techniques*. New York, NY, USA: Springer, 2010.
[31] M. Fowler, *UML Distilled: A Brief Guide to the Standard Modeling Object Language*, 3rd ed., series Object Technology Series. Reading, MA, USA: Addison-Wesley, Sep. 2003.
[32] M. La Rosa, W. van der Aalst, M. Dumas, and A. ter Hofstede, "Questionnaire-based variability modeling for system configuration," *Softw. Syst. Model.*, vol. 8, pp. 251–274, 2009.
[33] R. Mitkov and L. Ha, "Computer-aided generation of multiple-choice tests," in *Proc. Int. Conf. Natural Language Process. Knowl. Eng.*, 2003, pp. 17–22.
[34] A. Hoshino and H. Nakagawa, "A real-time multiple-choice question generation for language testing: A preliminary study," in *Proc. 2nd Workshop Building Educational Appl. Using NLP*, 2005, pp. 17–20.
[35] G. Gröner, M. Asadi, B. Mohabbati, D. Gašević, F. Silva Parreiras, and M. Bošković, "Validation of user intentions in process models," in *Proc. 24th Int. Conf. Adv. Inf. Syst. Eng.*, 2012, pp. 366–381.
[36] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., *The Description Logic Handbook*. New York, NY, USA: Cambridge Univ. Press, 2007.
[37] A. Queralt and E. Teniente, "Verification and validation of UML conceptual schemas with OCL constraints," *ACM Trans. Softw. Eng. Methodol.*, vol. 21, no. 2, pp. 13:1–13:41, 2012.
[38] C. Pons and D. Garcia, "A lightweight approach for the semantic validation of model refinements," *Electron. Notes Theor. Comput. Sci.*, vol. 220, no. 1, pp. 43–61, Dec. 2008.
[39] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray, " On challenges of model transformation from UML to alloy," *Softw. Syst. Model.*, vol. 9, pp. 69–86, 2010.
[40] D. Jackson, "Alloy: A lightweight object modelling notation," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 2, pp. 256–290, Apr. 2002.
[41] A. Van Lamsweerde, "Requirements engineering in the year 00: A research perspective," in *Proc. 22nd Int. Conf. Softw. Eng.*, 2000, pp. 5–19.
[42] B. Nuseibeh and S. Easterbrook, "Requirements engineering: A roadmap," in *Proc. Conf. Future Softw. Eng.*, 2000, pp. 35–46.
[43] N. A. M. Maiden and G. Rugg, "ACRE: Selecting methods for requirements acquisition," *Softw. Eng. J.*, vol. 11, no. 3, pp. 183–192, 1996.
[44] E. S. Cordingley, "Knowledge elicitation techniques for knowledge-based systems," in *Knowledge Elicitation: Principle, Techniques and Applications*, D. Diaper, Ed, New York, NY, USA: Springer-Verlag, 1989, pp. 87–175.
[45] E. Turban, *Decision Support and Expert Systems: Management Support Systems*, 3rd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 1993.
[46] K. A. Ericsson and H. A. Simon, *Protocol Analysis: Verbal Reports As Data*. Cambridge, MA, USA: MIT Press, 1993.
[47] G. Rugg, C. Corbridge, N. Major, A. Burton, and N. Shadbolt, "A comparison of sorting techniques in knowledge acquisition," *Knowl. Acquisition*, vol. 4, no. 3, pp. 279–291, 1992.
[48] C. Corbridge, G. Rugg, N. Major, N. Shadbolt, and A. Burton, "Laddering: Technique and tool use in knowledge acquisition," *Knowl. Acquisition*, vol. 6, no. 3, pp. 315–341, 1994.
[49] B. Selic, "What will it take? a view on adoption of model-based methods in practice," *Softw. Syst. Model.*, vol. 11, pp. 513–526, 2012.
[50] L. Garshol, "Metadata? thesauri? taxonomies? topic maps! making sense of it all," *J. Inf. Sci.*, vol. 30, no. 4, pp. 378–391, 2004.

Marco Autili is an assistant professor in the Department of Information Engineering, Computer Science, and Mathematics, University of L'Aquila, Italy. His main research areas are software engineering, formal methods, distributed systems, and context-oriented programming. His main research and development (R&D) activities include: automated synthesis of software connectors; formal specification and analysis of complex distributed systems; context-oriented programming and resource-oriented analysis of adaptable (mobile) applications. He published several papers in leading journals, conferences, and workshops. He is (has been) a member of many Program Committees and a reviewer of many journals among which *Science of Computer Programming*, *Software and Systems Modeling*, *J. of Systems and Software*, *Automated Software Eng.*, *J. of Internet Services and Applications*, *J. of Logic and Algebraic Programming*, and a number of Transactions. He is (has been) involved in many EU and Italian projects contributing to R&D, management and coordination activities. Please, visit http://www.di.univaq.it/marco.autili/.

Antonia Bertolino is a research director at CNR-ISTI, Pisa. Her research covers software and services engineering, with particular interest in testing approaches. She serves as the software testing area editor for the *Elsevier Journal of Systems and Software*, and is currently an associate editor of the *ACM Transactions on Software Engineering and Methodology* and of the *Springer Empirical Software Engineering Journal*. She is the general chair of the ACM/IEEE Conference ICSE 2015, to be held in Florence, Italy.

Guglielmo De Angelis received the PhD degree in industrial and information engineering from Scuola Superiore Sant'Anna of Pisa. He is a technologist at CNR-ISTI, temporary posted to CNR-IASI. His research area is software engineering, where he mainly focuses on model-driven engineering and on service-oriented architectures.

Davide Di Ruscio is an assistant professor in the Department of Information Engineering Computer Science and Mathematics, University of L'Aquila. His research interests are related to several aspects of model-driven engineering (MDE) including domain specific modeling languages, model transformation, model differencing, model evolution, and coupled evolution. He has coauthored more than 70 papers in various journals, conferences, and workshops on such topics. He has been involved in the organization of several international workshops and conferences, and reviewer of many journals like *Science of Computer Programming*, and *Software and Systems Modeling*. Since 2006, he has been involved in European projects mainly in the field of model-driven engineering, service based systems, and open source software. More information is available at http://www.di.univaq.it/diruscio.

Alessio Di Sandro received the degree (with honors) in computer engineering from the University of Pisa in 2009. During the masters thesis, he was also with Ericsson Research in Stockholm, Sweden. He was a researcher at CNR-ISTI in Pisa, Italy, and currently he is a researcher at the University of Toronto, Canada. His research interests include topics from model-driven engineering and visual technologies, such as model management, model validation, automated code generation, design of graphical interfaces, web technologies.