Streamlining scenario modeling with Model-Driven Development: a case study

Miguel Goulão, Ana Moreira, João Araújo, João Pedro Santos
Departamento de Informática, CITI, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Portugal
{miguel.goulao, amm, ja, joao.santos}@di.fct.unl.pt

Abstract—**Scenario modeling can be realized through different perspectives. In UML, scenarios are often modeled with activity models, in an early stage of development. Later, sequence diagrams are used to detail object interactions. The migration from activity diagrams to sequence diagrams is a repetitive and error-prone task. Model-Driven Development (MDD) can help streamlining this process, through transformation rules. Since the information in the activity model is insufficient to generate the corresponding complete sequence model, manual refinements are required. Our goal is to compare the relative effort of building the sequence diagrams manually with that of building them semi-automatically. Our results show a decrease in the number of operations required to build and refine the sequence model of approximately 64% when using MDD, when compared to the manual approach.**

Keywords–Scenario Modeling; Model Transformations; Model-Driven Engineering

I. INTRODUCTION

Scenarios [1, 2] are widely used in Requirements Engineering (RE) to represent paths of possible behavior of a use case. Approaches using UML [3] represent scenarios through activity [4] and sequence models [5]. While activity models are mostly used in the preliminary stages of software development process, sequence models tend to be used later, when detailed descriptions of object interactions become necessary.

Some behavioral and structural abstractions present in activity models can be reused automatically in sequence models by means of transformations. Petriu and Sun proposed a process to generate activity models from sequence models [6] in a reverse engineering approach, where the source model is more fine-grained than the generated model. This is useful when handling legacy systems. However, in a context where we first model the system at higher levels of abstraction and then progressively move towards a more fine-grained models, the solution proposed in [6] does not help. In [7] we discussed how we can use Model-Driven Engineering (MDE) techniques [8-10] to define transformations from activity to sequence models. The generated models then may be refined to add required details that are not present in activity models. Our goal was to decrease the effort involved in modeling scenarios. For this study, we assume that the activity model is correct and models the original use case faithfully. However, in a general case this may not be so, what would result in a need to remove of modify elements in the generated sequence diagram.

In this paper, our goal is to conduct a case study to assess the impact of our MDD approach to streamline scenario modeling, with respect to its effort of construction, and compare it with that of generating similar scenario models manually. As a surrogate for effort, we use the number of refinement operations (insertions and removals) performed while developing the models. Our case study uses 11 scenarios from the mobile media domain.

The models built following this MDD approach are also potentially easier to trace back to the activity models (with the help of the transformation rules) and are built using sequence models design best practices, although a detailed discussion of these benefits is outside of the scope of this paper.

The remaining of this paper is organized as follows. Section II outlines transformation rules to map activity models into sequence models and addresses the refinement of the generated models. Section III introduces the supporting tool to implement those transformations. Section IV illustrates our approach with a scenario of a case study. Section V compares the effort of modeling the scenario by hand with that of refining the generated model and Section VI discusses related work. Finally, Section VII concludes the paper and provides directions for future work.

II. MIGRATING FROM ACTIVITY TO SEQUENCE MODELS

This section summarizes the transformation rules and the refinements that can be applied to the generated model (for further details, see [7]). We assume that the activity models are deterministic.

A. Generating Sequence Models

Rule 1: Generating Objects in Sequence Models. Boundary and control objects are created by default in sequence models with the name of the activity model that represents the scenario under study. In activity models, it is common to represent access operations (*read* or *write*) to objects, with flows between activities and objects. We map objects found on activity models to entity objects in the sequence model. Actors in a sequence model are generated to match actor swimlanes on the activity model.

Rule 2: Generating Messages in Sequence Models. Each *activity* in an activity model is mapped into a message in the sequence model. Decomposable messages can then be refined into a set of messages. Our approach uses sub-rules to identify the source and target objects of the generated messages, i.e., which object is the *caller* and which is the *callee*.

- **Rule 2.1: Object flows.** The direction of the flow connecting an activity to an object indicates if it is a *read* or a *write* operation. For a *write* operation, a return message with type *void* from the entity to the control

object is created. *Read* operations require a return message with type **not *void***, which is created from the control to the entity object.

- **Rule 2.2: Message name.** Some message names implicitly give information about the objects' type (boundary, control, or entity). For example, a message with a name such as *showMessage()* is typically sent to boundary objects to display messages to the user.
- **Rule 2.3: Swimlanes.** When a message is generated from an activity that is inside an actor's swimlane, the source object of that message is of type actor. As actors only access interfaces, the pattern is that the source and target of the message are the actor and boundary objects, respectively.
- **Rule 2.4: Redirecting Messages.** Boundary objects redirect messages from actors to control objects, and vice-versa. Messages to achieve this goal are created automatically.

Rule 3: Generating Sequence Model Operators. Sequence models may use several kinds of fragments: **ALT** (or alternative), **PAR** (or parallel), **OPT** (optional), and **LOOP**. Each of these fragments is generated by sub-rules.

- **Rule 3.1: Generating *PAR* Operators.** *A PAR* operator is created in the sequence model for each pair of *fork-join* elements is in the activity model. The elements between *fork* and *join* are included in a *PAR* fragment.
- **Rule 3.2: Generating *ALT*, *OPT* and *LOOP* Operators.** Algorithms for graphs with cycle detection mechanisms can be used to detect cycles in an activity model. Activity models can be viewed as graphs, where activities and flows between activities are seen as nodes and edges, respectively. For each cycle detected, a *LOOP* operator is created with a guard condition, respective messages and sub-operators. If the number of output flows is 1, an *OPT* fragment is created with its guard condition. If the number of outgoing flows is greater than 1, a fragment *ALT* is created. Within this fragment, there should be an alternative for each outgoing flow with its guard condition. The elements inside the flow of each guard are moved to the respective fragment.

B. *Refining Sequence Models*

After generating the sequence model, the domain analyst must refine it. This is needed because sequence models are more fine-grained than activity models and, hence, additional information should be provided to the generated model. The modeler should follow these typical refinements: add arguments and types; decompose a message to a set of messages; add return messages; add variables; initialize guards; delete undesired elements.

III. Tool Support

We implemented a plug-in for the Eclipse platform [11] to support the transformations described in the previous section. We used the Eclipse Modeling Framework (EMF) and UML2 plug-in for Eclipse[1]. After this generation, the user can use the EMF environment to refine the sequence model.

A traceability metamodel is used to link abstractions from the activity to sequence models (Figure 1). This metamodel is composed of activity (left) and sequence (right) model elements. Metaclasses are used to link activity to sequence abstractions (center). The central abstractions unify the concepts present on activity and sequence models and reflect the result of the transformation. The metamodel element with name *Activity-Message* allows preserving the connection between activities and the sequence model messages that are generated. The element *Object–EntityObject* connects the objects found on the activity model and the generated entity object in the sequence model. Finally, the element *Swimlane-Actor* shows how swimlanes in the activity model are the source for the actor objects.

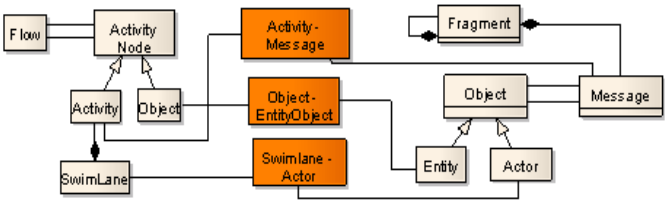

Figure 1 - Activities and Sequence Models Unified Metamodel

IV. Application to a Case Study

Mobile Media [12] is a software system offering operations on photos, music and videos on mobile devices. The use case model for this case study is shown in Figure 2.

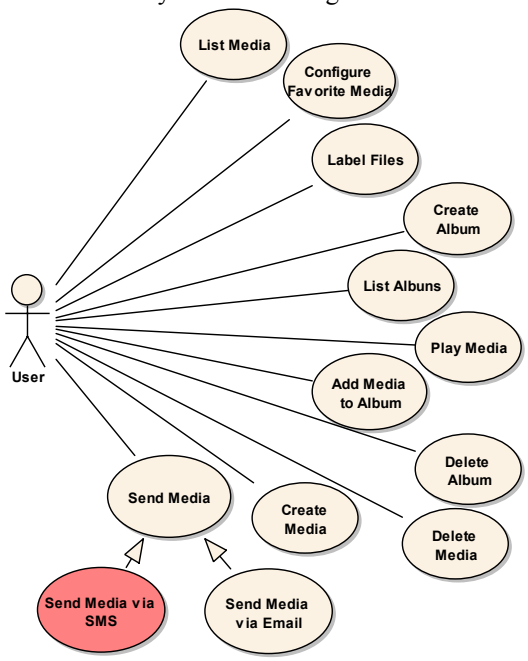

Figure 2 – Use case model for the Mobile Media

The user can manipulate data, such as adding and deleting media, configure a media file as a favorite, add or delete media albums. He can access the data on the device, list albums, media, view the favorites media or eventually play a media file

[1] www.eclipse.org/uml2/

(play a video, see a photo or hear a sound). Finally, the user can share the media data with other mobile media users, by sending messages. These messages can be sent via an SMS or Email protocol.

For illustration purposes consider the Send Media via SMS scenario (Figure 3). In *Send Media via SMS*, the user starts by selecting the Send Media via SMS option, then the system asks for media to send. The user selects the media to send in the message. Then, he specifies the target number of the message. This information is enough to send the message to the target mobile device (activity Send Message). If the message is sent without errors, it is saved locally in the Mobile Media system and *"Message Sent Successfully"* is shown to the user.

We generate the sequence model depicted in Figure 4 for the scenario Send Media via SMS applying the rules discussed in Section II-A. To illustrate the transformation process, some elements in Figure 4 contain a red numbered circle to denote part of the transformation rules that were applied.

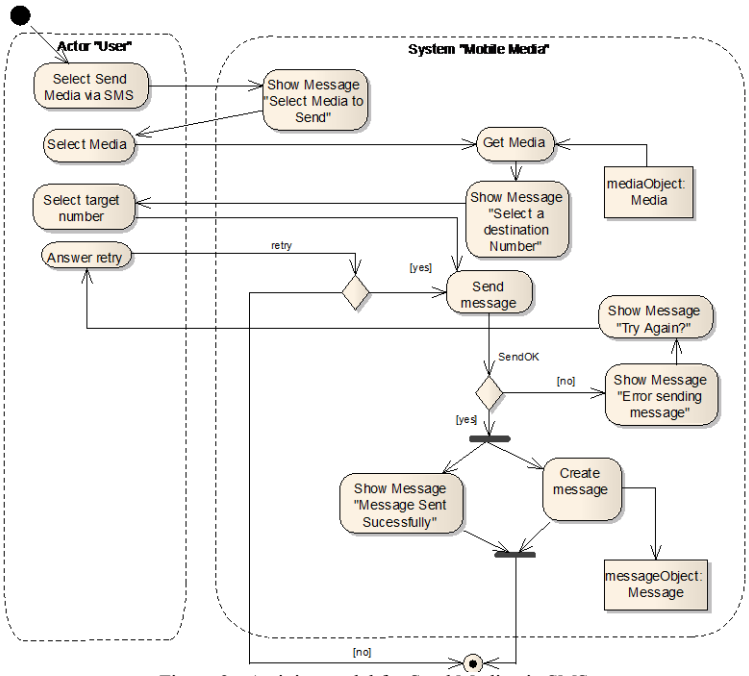

Figure 3 - Activity model for Send Media via SMS

Table I shows how each numbered element was created and the corresponding transformation rule. After the generation of the candidate sequence model, some refinements must be done to improve it. The following shows some of the possible refinements for this example:

- The message *selectMedia()* can be completed with an argument of type *String*, denoting the path of the selected media.
- The message *selectTargetNumber()* can be completed with an argument of type *integer*, denoting the destination number of the message.
- The message *sendMessage()* can be completed with two arguments: the path of the selected media and the destination number of the message. The return of that message should also be assigned to a variable *sendError* which will be evaluated on the *LOOP* operator.
- The variable *retry* of the loop fragment must be initialized to be evaluated on the first iteration of the loop. In this case, the value should be *retry = yes* in order to execute the loop the first time. The *answerRetry()* return value should also be assigned to the *retry* variable.

The result of the refinements is presented in Figure 5.

TABLE I. RULES APPLIED FOR SEQUENCE MODEL GENERATION

Number	Rule Applied
1	**Rule 2.2.** This message was created from the activity *Select Send Media Via SMS*. The source of the message is the actor object, since it was the first generated message.
2	**Rule 2.4.** This message was created using the rule that redirects a message from the *actor* object to the *control* object.
3	**Rule 2.3.** This message has *interface* as the target object since the message name fits with the pattern *ShowMessage*
4	**Rule 2.1.** This message was created with name *Media* since the last created message denotes a read operation.
5	**Rule 2.2.** The name of this message was derived from an activity with the same name. This message has the *control* object as source and target, since no other rules were applicable in this situation.
6	**Rule 3.2.** This fragment was created as a loop was detected in a decision node with an outgoing flow with guard *[sendOK = no]*. Since the cycle includes also the outgoing flow with guard *[yes]* both conditions must be true to enter the loop fragment.
7	**Rule 3.2.** This fragment was created since a decision node with two outgoing flows and no loops were detected on the activity model.
8	**Rule 2.1.** This message was created with type *void* since the previously created message denotes a write operation.

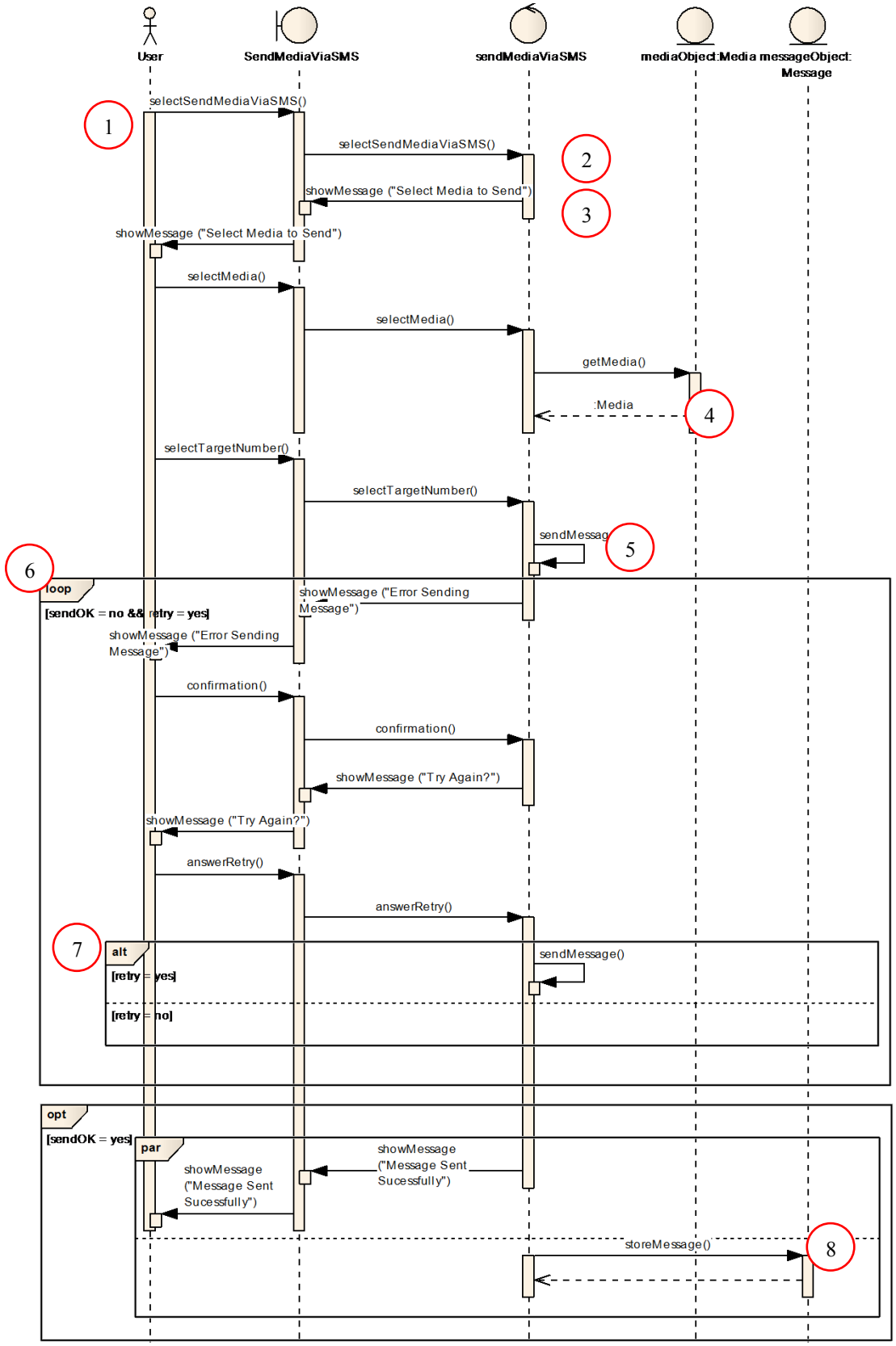

Figure 4 - Sequence model for the scenario Send Media Via SMS

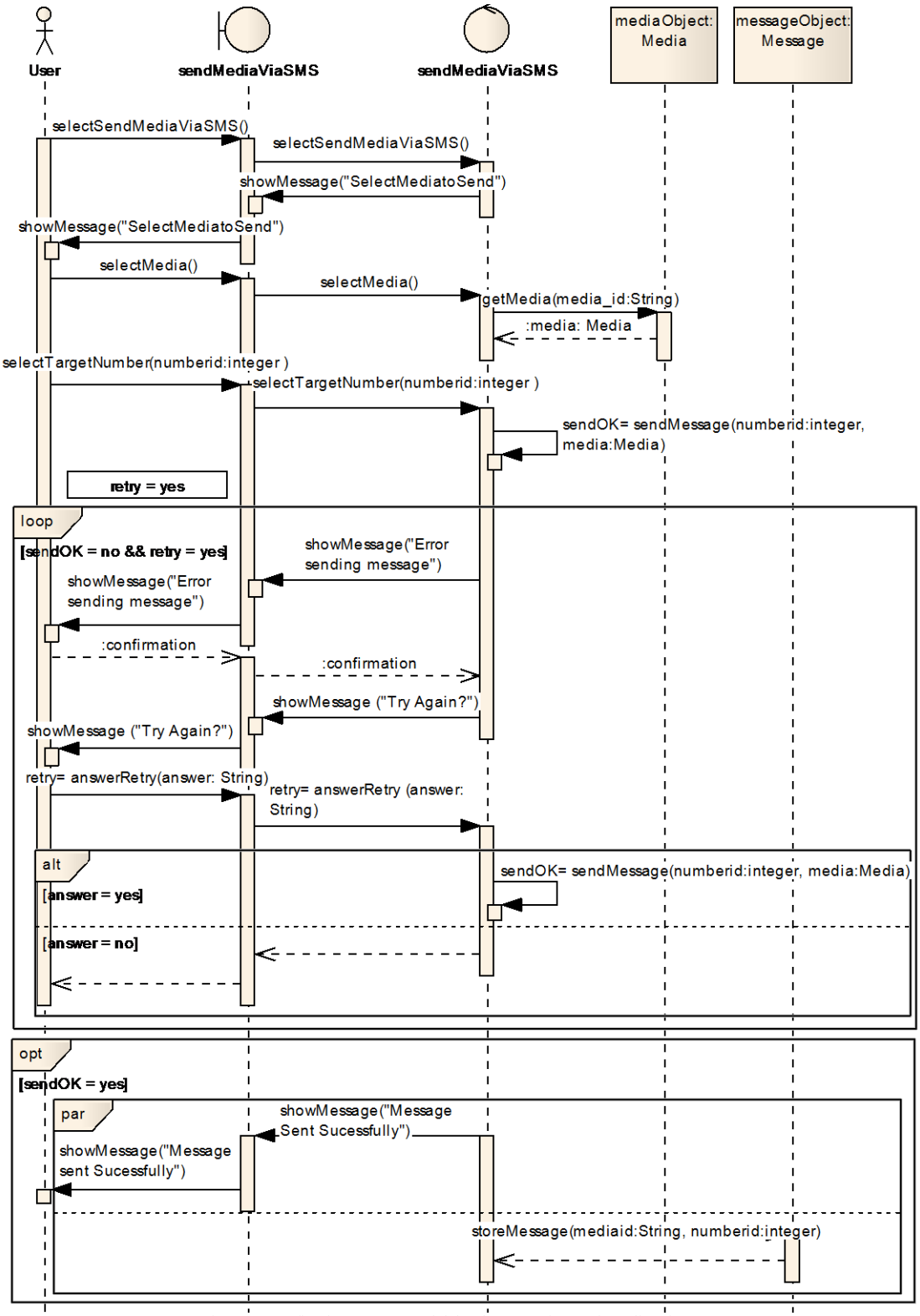

Figure 5 - Refined sequence model for the scenario Send Media Via SMS

V. VALIDATION AND DISCUSSION

In this section, our goal is to analyse sequence models built with and without our sequence model generation approach, for the purpose of comparing model construction, with respect to the relative effort, from the point of view of modellers, in the context of the Mobile Media case study.

Our research hypothesis is that *the usage of our approach allows a significant effort reduction, when compared to building sequence models from scratch*. To compare the effort of creating a sequence model from scratch with that of using our approach to generate a model and then refining it, we associate each sequence model element action with a cost. Consider the following actions made in a sequence model:

(i) removal of any kind of element;
(ii) insertion of a variable/argument name;
(iii) insertion of a variable/argument type;
(iv) insertion of an operator (PAR, ALT, etc.) and respective guard conditions;
(v) insertion of an object and its name;
(vi) insertion of a message and the corresponding procedure call name (if necessary).

For simplicity, assume all these actions have a similar time cost and assign one unit of time cost to each of them. If we were to build the model presented in Figure 5 from scratch, this would require 72 editions. In contrast, if we start by generating the model in Figure 4 and then apply a sequence of editions, we only need 32 editions (30 additions + 2 removals) to obtain the model in Figure 5. The effort has decreased from 72 to 32 editions, which corresponds to a reduction of around 55%. As part of our validation effort, more scenarios were developed and sequence models generated successfully, in the context of the Mobile Media case study. To simplify, we consider all types of action as having the same cost. Table II summarizes this information, showing for each scenario:

(i) *number of elements* that the sequence model contains;
(ii) *number of insertions* necessary during refinement;
(iii) *number of deletions* performed during refinement;
(iv) total *number of refinements* (insertions + removals).

If we consider a model with *n* elements, the assumption is that the effort for building that model is the effort for making *n* editions, corresponding to the *n* additions required for building the model, when we are starting with an empty model. Note that we assume that the effort for any extra edition to the model is fixed, so that the overall effort is linearly proportional to the number of editions.

The effort to create the listed sequence models from scratch is 270 editions (the number of elements), while the total cost to refine them is 97 (the number of refinements). The effort has decreased from 270 to 97 editions, a value that shows a significant improvement of around 64%. We can also observe that most refinement actions are insertions (88) rather than deletions (9). This means that most of the automatically generated elements are correct. Additional edits to the generated models are dominated by insertions, with relatively few deletions. In fact, 5 of the scenarios required no deletions at all.

TABLE II. NUMBER OF ELEMENTS AND REFINEMENTS FOR EACH SCENARIO

Scenarios	Number of Elements	Number of Insertions	Number of Removals	Number of Refinements
Send Media via SMS	**72**	30	2	**32**
Send Media via Internet	**74**	31	2	**33**
Create Media	**20**	4	1	**5**
Delete Media	**8**	2	0	**2**
Create Album	**20**	4	1	**5**
Add Media to Album	**32**	6	2	**8**
List Media	**8**	2	0	**2**
Configure Favourite Media	**10**	2	1	**3**
List Albums	**8**	2	0	**2**
Play Media	**10**	3	0	**3**
Delete Album	**8**	2	0	**2**
Total	**270**	88	9	**97**

Table III presents descriptive statistics on our sample, namely the *number of scenarios*, *mean*, *standard deviation*, *minimum* and *maximum* values, *skewness* and *kurtosis*, for *elements*, *insertions*, *deletions*, and *refinements* (i.e. insertions and deletions). The relevant metrics for testing our research hypotheses, emphasized in **bold** in table II, are the number of elements in a sequence model designed from scratch (*Elements*), and the total number of refinements required for building a sequence model starting from the model generated with our approach (*Refinements*). The former represents the effort in the baseline approach to build sequence models, while the latter represents the effort of refining sequence models from the models built using our approach.

TABLE III. DESCRIPTIVE STATISTICS

Metric	N	Mean	SDev	Min	Max	Skew	Kurt
Elements	**11**	**24.55**	**25.125**	**8**	**74**	**1.570**	**1.099**
Insertions	11	8.00	11.198	2	31	1.869	1.916
Deletions	11	0.82	0.874	0	2	0.409	-1.621
Refinements	**11**	**8.82**	**11.856**	**2**	**33**	**1.821**	**1.791**

In order to test our research hypothesis, we start by specifying it in terms of a null and an alternative hypothesis.

H0: *The two samples come from identical populations, meaning that our approach does not reduce time costs in scenario modeling.*

H1: *The two samples come from different populations, meaning that our approach reduces time costs in scenario modeling.*

The first thing to check is whether or not our two samples have a normal distribution. The descriptive statistics hint to an asymmetric (right-skewed) distribution in both samples and, therefore to a non-normal distribution.

Table IV summarises the results of two normality tests: the Kolmogorov-Smirnov, with Lilliefors Significance correction [13], and the Shappiro-Wilk tests [14]. The null hypothesis in both tests is that the samples have a normal distribution. The alternative is that the samples have a non-normal distribution. Both tests confirm that the distributions of Elements and Refinements are not normal, with $p < 0.01$.

TABLE IV. NORMALITY TESTS

	Kolmogorov-Smirnov			Shapiro-Wilk		
Metric	**Statistic**	**df**	**Sig.**	**Statistic**	**df**	**Sig.**
Elements	0.229	11	0.007	0.690	11	0.000
Refinements	0.354	11	0.000	0.608	11	0.000

The outcome of the normality tests leads us to use a non-parametric test to test our research hypothesis. We will use the Wilcoxon Signed Rank test [15] to determine whether the differences among added elements and refinements of our paired samples are statistically significant. This test is a non-parametric alternative to the paired samples Student's t-test [16]. The null hypothesis is that the number Elements and Refinements could come from the same sample. The alternative is that they come from different samples.

Table V presents the output of this test's ranks. The differences are separated into 3 groups: Negative Ranks (Refinements < Elements), Positive Ranks (Refinements > Elements) and Ties (Refinements = Elements). Note that all cases present negative ranks. In other words, in all cases we have fewer Refinements than Elements.

TABLE V. RANKS

		N	Mean Rank	Sum of Ranks
Refinements - Elements	Negative Ranks	11	6.00	66.00
	Positive Ranks	0	0.00	0.00
	Ties	0		
	Total	11		

Table VI presents the test statistics. With $p = 0.003 < 0.01$, we can reject the null hypothesis and accept the alternative.

TABLE VI. TEST STATISTICS

	Refinements-Elements
Z	-2.952
Asymp. Sig. (2-tailed)	0.003

In summary, the results seem to indicate that our approach decreases costs (mean rank of 0.00 vs. mean rank of 6.00). The Wilcoxon signed rank test shows that the observed difference between both measurements is significant. Thus we can reject the null hypothesis that both samples are from the same population, and we may assume that using our approach leads to a significantly lower cost in building sequence diagrams, when compared to the alternative of building them from scratch.

Figure 6 further illustrates our point. It depicts a graph where, for each scenario, the number of elements that it is composed of is represented by the vertical coordinate (it reflects the number of actions needed to construct the scenario model from scratch) and the number of actions needed to refine the scenario when our approach is used is represented as the horizontal coordinate. Figure 6 also shows the line that best fits the drawn points. We can see that the different scenarios differ substantially in terms of their number of elements. Functionalities such as *List Media, List Albums* and *Delete Media,* are simple functionalities of the mobile media. Other functionalities such as *Send Media via SMS* and *Send Media via Internet* are functionalities that involve more objects and communications (through messages) among them.

All the points are relatively close to the line that best fits these points, meaning that this line characterizes well the relationship between the number of model elements of a scenario, and the number of elements needed to refine in the case our approach is applied. We could analyze these points with more detail by applying theory of linear regression. However, it is visually clear that the line fits well with the observations. There is a linear relationship between the number of model elements of a scenario and the number of actions needed to refine that scenario when our approach is applied. Indeed Elements and refinements have a Spearman correlation of 1.000, significant at 0.01 level (2-tailed). The slope of this line also shows that as the number of elements of a scenario increases, the number of refinements also increases linearly. However, the number of refinements needed is only a percentage of the total number of the scenario model elements (around 64%, as we saw previously). Having a linear increase means that our approach is also scalable or, in other words, that the relative number of refinement actions does not increase unexpectedly as the number of scenario model elements increases.

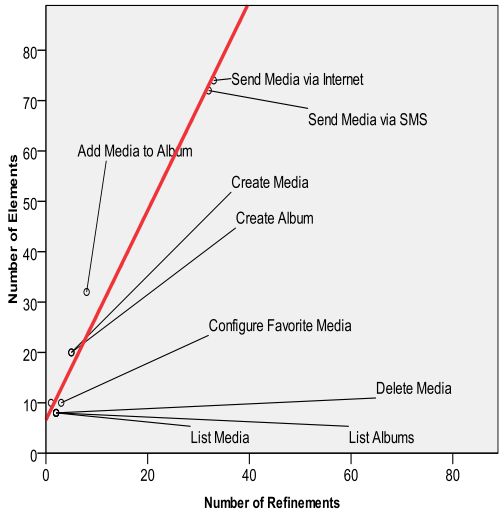

Figure 6 - Graphic showing Number of Elements vs Number of Refinements

In any experimental-based validation effort, one must always consider potential threats to validity, limiting our capacity of generalization of the statistical test results for all the cases, all the software domains, all the scenarios.

The statistical tests presented previously, were performed on sequence diagrams produced in the context of the Mobile Media system, which is a system that interacts intensively and mainly with the user to access and write data. This means that

the interfaces, processes and data tiers of each scenario are well established and thus, sequence models are good models to define and represent bounds. In the context of other domains or scenarios, if these bounds are not so clear, the results can be different. Further experimentation in those domains would be required to assess this approach on those domains.

Another threat to validity is related to the fact that, in order to obtain such good results, the names of the activities should follow some patterns, to enable using the set of transformation rules defined. Objects in activity diagrams should also be defined to represent readings and writings since some of the transformation rules need this information to be present in the model.

The usage of a non-weighted effort unit for insertions and deletions which is agnostic to whether these are made in the context of a model built from scratch, or in the context of a model refinement is also a potential threat. This simplification should not affect significantly the outcome of this validation, as deletions represent less than 10% of all the refinements performed upon the generated sequence models. Nevertheless, further research is required to calibrate the efforts of deletions, in contrast with those of insertions, as well as the impact of building the model from scratch *vs.* refining a model generated with our MDD transformations.

A related issue concerns the relative weight of editing a model built from scratch when compared to editing a generated model. When editing a generated model (or, for that matter, a model built by someone else), we might also want to consider the time invested in understanding the existing model, before making changes to it. Again, further research is required to assess the extent to which that effort is significantly different from the one spent when editing a model built manually.

One important limitation of our approach is related to the reuse of refinements performed by the user when the sequence model is re-generated. The refinements done previously are currently lost and must be redone by the modeller. We are now working to support reuse of refinement information as a future step.

VI. Related Work

In [17], the authors present an algorithm to automatically generate UML statecharts from a collection of scenarios represented using UML sequence models. In this work, they address several issues, such as detecting conflicts arising from the merging of independently developed sequence models and find behavioural similarities between different sequence models. They do this at the algorithm or transformation level.

In [18], Activity Models are extended with process goals and performance measures to make them conceptually visible. They also provide transformation rules to BPEL (Business Process Execution Language) to make the measures available for execution and monitoring. In this work, the additional notation defined, for the activity models, allow both models being semantically closer, which made the definition of the transformation rules easier.

Finally, in [19] the authors propose to generate automatically, through model transformations, an activity model representing the use case scenario from a textual template. In this work, we observed that the semantics inherent to the abstractions present on the template (if-then-else, requirements numeration indicating parallelism) and on the activity model were very close, which resulted in a relatively trivial set of transformation rules.

In our approach, we have not extended the activity and sequence models standard notation; we concentrate our effort on the definition of transformation rules to facilitate the semi-automatic generation of sequence models from activity models. Both models have different levels of granularity, representing different system perspectives, which makes the definition of transformation rules more difficult. However, since some information between them overlaps, such as, for example, conditional behaviour or concurrency, it is possible to automate part of the process using model transformations.

The evaluation performed in this case study uses the number of editions made in a sequence diagram as a surrogate for the effort required for building it. In the case of the sequence models built manually, this corresponds to the number of model elements used in the model. The idea of using model elements as surrogates for effort has been used in several occasions. For example, Use Case Points have been used to predict the development effort for software systems [20]. This approach was influenced by a previous approach for effort prediction (Function Points Analysis [21]). These are two of the many examples in the literature where a high level view of a software system is used for estimating the effort required for building it. In that sense, both Use Case Points and Function Points are also used as surrogates for effort.

As noted in section V, our assessment of the quality of the generated models is that the vast majority of the generated model elements are kept during model refinement, which suggests a high quality of the model transformation. Indeed, the low number of element removals during model refinement is an indicator of the quality of the model generator. In a nutshell, we are able to generate a significant portion of the model, with few mistakes. There are other possible perspectives on model transformation quality, not followed in this paper. For example, the quality of the input model can be contrasted with that of the output model [22], although this is not applicable for our context, as the input and output models are at different abstraction levels. Another perspective is to assess the quality of the model transformation rules themselves, as proposed in [23] and [24]. Again, these metrics are not directly helpful for our research question (essentially, can we decrease the effort in model building without sacrificing the quality of the final model?), as they focus on the quality of the transformation rules, from a complexity perspective. The same applies to metrics-based approaches to assessing the efficiency of model transformations [25]. While complexity and efficiency of model transformations are relevant concerns, we are more interested in the deliverables of the transformation process, and how they can help us speeding up modeling activities.

VII. Conclusions and Future Work

Modeling scenarios with activity and sequence models of a system can be semi-automated by using transformation techniques, a key concept in MDE. By using transformations, it is possible to reuse abstractions that were directly mapped from one model to another. This frees the burden of the modeller from creating similar abstractions that can be automatically generated and also avoids modeling errors, concentrating the effort on the refinement stage of generated artefacts. Transformation rules were defined to generate sequence models artefacts from activity models artefacts. Our transformational rules support the automation of the creation of objects, messages and operators for sequence models from the information contained in activity models.

Our initial validation effort, through the case study, described in Section IV, provided encouraging feedback concerning the desired effort reduction. Indeed, the number of edits required for building a sequence model from the activity model decreased by around 64%, when using our semi-automatic transformation approach. The advantages, from a quality point of view, include: (i) a reduction in the effort building the sequence model, (ii) increased traceability among models (through the semi-automatic translation rules), (iii) error prevention when migrating from different scenarios notations, and (iv) support for reuse of sequence models design best practices, thus providing a good stepping stone for high quality scenario modeling.

For future work, we plan to implement the transformation rules described in this paper, and apply our approach in projects where real case studies are available in order to further validate the claim about time costs improvement provided by our approach. Finally, we plan to extend our approach to support reuse of refinement information.

Acknowledgment

The research described in this paper was partially funded by European AMPLE Project IST-33710 and CITI – PEst - OE/EEI/UI0527/2011, Centro de Informática e Tecnologias da Informação (CITI/FCT/UNL) - 2011-2012) – for the financial support for this work.

References

[1] I. F. Alexander and N. Maiden, *Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle*: John Wiley and Sons, ISBN: 0-470-86194-0, 2004.

[2] M. Karen and H. Karan, *User-centered requirements: the scenario-based engineering process*: Lawrence Erlbaum Associates, Inc., ISBN: 0-8058-2065-5, 1997.

[3] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*: Addison Wesley Longman Inc., ISBN: 0-201-57168-4, 1998.

[4] M. Alférez, U. Kulesza, A. Sousa, J. Santos, A. Moreira, J. Araújo, and V. Amaral, "A Model-Driven Approach for Software Product Lines Requirements Engineering," Proc. 20th International Conference on Software Engineering and Knowledge Engineering, Knowledge Systems Institute Graduate School, San Francisco Bay, USA, July 1-3, 2008, pp. 779-784.

[5] W. Hongyuan, Z. Ke, F. Tie, C. Haiyan, and Z. Yinshi, "Synthesizing Statecharts Through Sequence Diagrams Analysis," Proc. Conference on Software Engineering and Applications, IASTED/ACTA Press, 2004, pp. 617-622.

[6] D. C. Petriu and Y. Sun, "Consistent behaviour representation in activity and sequence diagrams," Proc. UML 2000 - The Unified Modeling Language. Advancing the Standard: Third International Conference, Springer Berlin / Heidelberg, York, UK, October 2-6, 2000, pp. 359-368, doi: 10.1007/3-540-40011-7_27.

[7] J. P. Santos, A. Moreira, J. Araújo, and M. Goulão, "Increasing Quality in Scenario Modelling with Model-Driven Development," Proc. 7th International Conference on the Quality of Information and Communications Technology (QUATIC'2010), IEEE Computer Society, Porto, Portugal, September 29 - October 2, 2010, pp. 204-209, doi: DOI 10.1109/QUATIC.2010.36.

[8] M. Volter and T. Stahl, *Model-Driven Software Development*. Glasgow, UK: Wiley, ISBN: 0-470-02570-0, 2006.

[9] S. Beydeda, M. Book, and V. Gruhn, *Model-Driven Software Development*. Berlin, Germany: Springer, ISBN: 354025613X, 2005.

[10] S. J. Mellor, K. Scott, A. Uhl, and D. Weise, *MDA Distilled Principles of Model-Driven Architecture*. Boston, MA, USA: Addison-Wesley, ISBN: 0-201-78891-8, 2004.

[11] D. Carlson, *Eclipse Distilled*: Addison-Wesley Professional, ISBN: 978-0321288158, 2005.

[12] T. Young, "Using AspectJ to Build a Software Product Line for Mobile Devices," Master of Science Dissertation, advisor: G. Murphy, The University of British Columbia, 2005, doi: 10.1.1.94.9977.

[13] H. W. Lilliefors, "On the Kolmogorov-Smirnov test for normality with mean and variance unknown," *Journal of the American Statistical Association,* vol. 62, No. 318, pp. 399-402, June, 1967.

[14] S. S. Shapiro and M. B. Wilk, "An Analysis of Variance Test for Normality (Complete Samples)," *Biometrika,* vol. 52, No. 3/4, pp. 591-611, December, 1965, doi: 10.1093/biomet/52.3-4.591.

[15] F. Wilcoxon, "Individual Comparisons by Ranking Methods," *Biometrics Bulletin,* vol. 1, No. 6, pp. 80-83, December, 1945.

[16] D. W. Zimmermann, "A Note on Interpretation of the Paired-Samples t Test," *Journal of Educational and Behavioral Statistics,* vol. 22, No. 3, pp. 349-360, September, 1997, doi: 10.3102/10769986022003349.

[17] J. Whittle and J. Schumann, "Generating Statechart Designs from Scenarios," Proc. International Conference on Software Engineering, ACM, Limerick, Ireland, June 4-11, 2000, pp. 314-323, doi: 10.1145/337180.337217.

[18] B. Korherr and B. List, "Extending the UML 2 Activity Diagram with Business Process Goals and Performance Measures and the Mapping to BPEL," Proc. 25th International Conference on Conceptual Modeling (ER), Springer, Tucson, AZ, USA, November 6-9, 2006, pp. 7-18, doi: 10.1007/11908883_4.

[19] J. J. Gutiérrez, C. Nebut, M. J. Escalona, M. Mejías, and I. M. Ramos, "Visualization of Use Cases through Automatically Generated Activity Diagrams," Proc. Model Driven Engineering Languages and Systems (MoDELS 2008), Springer-Verlag Berlin Heidelberg, Toulouse, France, September 28 - October 3, 2008, pp. 83-96, doi: 10.1007/978-3-540-87875-9_6.

[20] B. Anda, H. Dreiem, D. I. K. Sjøberg, and M. Jørgensen, "Estimating Software Development Effort Based on Use Cases - Experiences from Industry," Proc. «UML» 2001 — The Unified Modeling Language. Modeling Languages, Concepts, and Tools. Fourth International Conference, Springer-Verlag Berlin Heidelberg, Toronto, Canada, 1-5 October, 2001, pp. 487-502, doi: 10.1007/3-540-45441-1_35.

[21] A. J. Albrecht, "Measuring Applications Development Productivity," Proc. IBM Applications Development Division Joint SHARE/GUIDE Symposium, Monterey, CA, USA, 1979, pp. 83-92.

[22] M. Saeki and H. Kaiya, "Measuring Model Transformation in Model Driven Development," Proc. CAiSE Forum, CEUR Workshop Proceedings, 2007.

[23] A. Vignaga, "Metrics for Measuring ATL Model Transformations," Universidad de Chile Technical Report TR/DCC-2009-6, 2009.

[24] M. F. van Amstel, "The Right Tool for the Right Job: Assessing Model Transformation Quality," Proc. IEEE 34th Annual Computer Software and Applications Conference Workshops, IEEE Computer Society, Seoul, Korea, July 19-23, 2010, pp. 69-74, doi: 10.1109/COMPSACW.2010.22.

[25] M. F. van Amstel, S. Bosems, I. Kurtev, and L. F. Pires, "Performance in Model Transformations: Experiments with ATL and QVT," Proc. Theory and Practice of Model Transformations, 4th International Conference, ICMT 2011, Springer Berlin / Heidelberg, June 27-28, 2011, pp. 198-212, doi: 10.1007/978-3-642-21732-6_14.