
Proceedings Paper:
Kolovos, Dimitrios S., Matragkas, Nicholas Drivalos, Williams, James R. et al. (2014) Model Driven Grant Proposal Engineering. In: Model-Driven Engineering Languages and Systems - 17th International Conference, MODELS 2014, Valencia, Spain, September 28 - October 3, 2014. Proceedings. , pp. 420-432.

https://doi.org/10.1007/978-3-319-11653-2_26

Model Driven Grant Proposal Engineering

Dimitrios S. Kolovos, Nicholas Matragkas,
James R. Williams, and Richard F. Paige

Department of Computer Science, University of York,
Deramore Lane, York, YO10 5GH, UK.
{dimitris.kolovos, nicholas.matragkas,
james.r.williams, richard.paige}@york.ac.uk

Abstract. We demonstrate the application of Model Driven Engineering techniques to support the development of research grant proposals. In particular, we report on using model-to-text transformation and model validation to enhance productivity and consistency in research proposal writing, and present unanticipated opportunities that were revealed after establishing an MDE infrastructure. We discuss the types of models and the technologies used, reflect on our experiences, and assess the productivity benefits of our MDE solution through automated analysis of data extracted from the version control repository of a successful grant proposal; our evaluation indicates that the use of MDE techniques improved productivity by at least 58%.

1 Introduction

The majority of experience reports in the field of Model Driven Engineering come from adopters in the software development industry and typically involve modelling and generating software. Here, we report on the use of Model Driven Engineering techniques in the context of research grant proposal development. Proposing and running collaborative research projects is one of the main activities undertaken by academics, and in our experience, existing tooling and processes for supporting some steps of this activity are sub-optimal. In this paper, we describe how we used MDE techniques to automate some of the laborious and error-prone steps of this activity, and report on the delivered productivity benefits, which we have measured through automated analysis of data from the version control repository of a successful proposal.

The rest of the paper is organised as follows. In Section 2, we outline the process of developing grant proposals and highlight its laborious and error-prone steps. In Section 3 we present how we applied Model Driven Engineering techniques to automate these steps, and discuss some key decisions we had to make along the way, and in Section 4 we present some measurements that demonstrate the obtained productivity improvements, and reflect on our experiences. In Section 5 we discuss related work and in Section 6 we conclude the paper.

2 Background

Often, a research project commences with the formation of a consortium comprising several academic and industrial partners, and is followed by the collaborative development of a grant proposal that outlines the objectives, technical organisation, and management of the project. In particular, the technical work needs to be decomposed into a number of work packages consisting of specific tasks and deliverables. Multiple partners can contribute to each work package and each partner can lead on the preparation and production of multiple deliverables.

Typically, proposal documents need to adhere to a template provided by the funding body, which prescribes their structure and formatting. Often, such templates require different views of the same information that appears in multiple places in the proposal (e.g. effort per work package, effort per partner, deliverables per work package, project deliverables ordered by delivery date). As a result, consistency of the proposal is an issue: if, for example, the effort associated with producing all of the deliverables in a project does not match the effort allocated to all partners in the project, there is a consistency problem (and such problems may be reflected in the proposal's review scores). To make matters worse, the information that could lead to inconsistencies in the proposal may change frequently during the core stages of development (e.g. the effort allocated to a partner for a particular work package may change several times during negotiations, or the deadline for a deliverable may be modified several times). Changes to this kind of information may require some substantial effort to implement in the proposal. For example, updating the due date of a deliverable in a proposal under the European Commission's 7th Framework Programme[1] template, requires updates in two separate tables (deliverables by chronological order, project Gantt chart), and in the section of the proposal that describes the deliverable itself. Similarly, modifying the effort of a partner in a work package requires updates in two different tables (effort per partner, effort for work package). In our experience, this information will change many times during the life-cycle of the proposal, and maintaining these views in a consistent state manually is both tedious and error-prone.

3 Model Driven Grant Proposal Engineering

In the spirit of MDE – that is, in the spirit of automating repetitive and error-prone tasks through the use of models and automated model processing – and in order to reduce the accidental complexity involved in developing grant proposals, we decided to use MDE techniques to automatically generate different views (tables, graphs, Gantt charts) of core proposal information. The generation process would be based on an abstract model of important information, and as a result the different views would be guaranteed to be consistent by construction. In this section we discuss our operating context, the approach we followed, and the challenges and opportunities we encountered along the way.

[1] `http://cordis.europa.eu/fp7/home_en.html`

3.1 Context

We focus on the process of developing proposals for EC-funded, ICT[2]-focused, collaborative, targeted research projects (STREP), however, developing proposals for other types of collaborative projects and research funding bodies should not be too dissimilar. Such proposals typically take between 3-6 months to prepare and involve 6-9 partner organisations from academia and industry, each participating with at least one representative in the proposal development phase. Proposals themselves range from 70-120 pages in length, and include technical, management and financial sections; technical and financial sections are most likely to change frequently during proposal development, whereas management sections tend to be reasonably stable. Partner representatives involved in the preparation of such projects typically have a computer science background or at least above-average computing skills. Our typical setup for collaborative development of such proposals involves a Subversion version control repository to which all partner representatives have read/write access, and which hosts the proposal document, distributed across many smaller LaTeX files in order to minimise merge conflicts.

3.2 Approach

Modelling To improve the internal consistency of the proposal and automate the repetitive steps of the proposal development process, we followed a bottom-up iterative approach in which we used an XML document to model the proposal we were working on at the time. We chose plain XML instead of a rigorous modelling framework, such as the Eclipse Modelling Framework [1], primarily due to XML's agility; using XML, we would be able to engage in exploratory modelling without being constrained by a rigid *metamodel*, and we would also not need to engage in metamodel-model co-evolution activities. On the other hand, by choosing XML we would miss strong typing and built-in support for cross-references between model elements – which we considered to be a fair trade-off in the context. Another agile option would have been to use an annotated general-purpose diagram [2] however, we considered XML to be more suitable as we anticipated that the information we would need to capture in the proposal model would have a predominantly hierarchical (as opposed to graph-based) structure. After a few iterations, we converged to a first version of the XML document that captured the work packages, tasks, deliverables, and milestones of the project (see Figure 1 for a *conceptual* metamodel to which project models conform to, and Listing 1.1 for a sanitised excerpt from a successful[3] proposal).

[2] Information & Communication Technology
[3] `http://www.ossmeter.org`

```
<?xml version="1.0"?>
<project name="OSSMETER" duration="30"
 title="Automated Measurement and Analysis of Open-Source
     Software" >

 <wp title="Requirements & Use Cases" leader="TOG" type="RTD">
  <effort partner="TOG" months="6"/>
  <effort partner="York" months="6"/>

  <task title="Use Case Analysis"
   start="1" end="6" partners="TOG, York, ..."/>
  <deliverable title="Project Requirements"
   due="6" nature="R" dissemination="PU" partner="TOG"/>
 </wp>

 <milestone title="Requirements and case studies completion"
     month="6"/>
 <milestone title="Project completion" month="30"/>

 <partner id="York" name="University of York"
  country="United Kingdom"/>
</project>
```

Listing 1.1. Sanitised excerpt of the model of the OSSMETER project

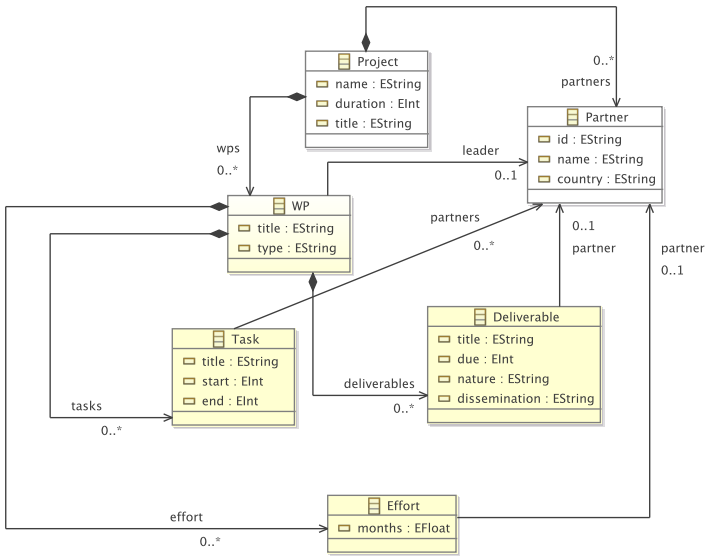

Fig. 1. Conceptual metamodel for project models

Model-to-text Transformation Our next activity was to decide how to best integrate any LaTeX content that we would generate from the constructed proposal model, with the hand-crafted parts of the proposal. We considered two options. The first option was to mix generated and hand-crafted content and rely on the M2T transformation engine's capabilities to preserve hand-crafted content upon re-generation. The second option was to keep generated and hand-crafted content separate, by producing a single file that would contain a number of auto-generated LaTeX commands which we could then reference from the hand-crafted LaTeX files. The main advantage of the first option was that contributors would not need to memorise any generated LaTeX commands; the main advantage of the second option was that we would enable contributors to use the generated LaTeX-commands in arbitrary locations in the proposal, without needing to adapt the generator every time. Automated content assistance and previewing facilities in modern LaTeX editors partially compensate for the shortcomings of the second approach, so as illustrated in Figure 2, we chose to produce a single LaTeX file containing a set of generated commands, which could then be imported by the main proposal file.

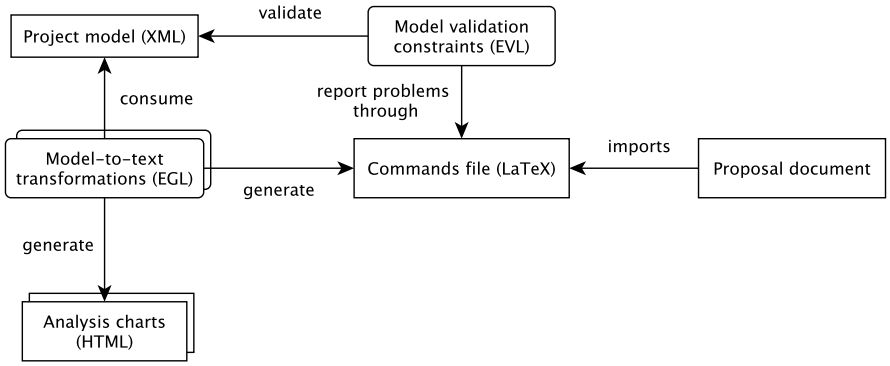

Fig. 2. Overview of the organisation of our MDE solution

In terms of the actual M2T transformation that would generate the LaTeX commands file, we chose to implement it using the Epsilon Generation Language [3], both because of prior familiarity with it but also because it provides built-in support for consuming plain XML documents in an elegant manner [4]. For example, the excerpt of the M2T transformation[4] presented in Listing 1.2, iterates through all work package (*wp*) elements in the XML document presented in Listing 1.1 (*t_wp.all* in line 2) and then through their *task* children (*wp.c_task* in line 5) to generate a LaTeX command (*workPackageAndTaskList* in line 1) that presents

[4] The complete M2T transformation is 529 lines long.

the project's work packages and tasks[5]. The executable content of an EGL template is contained within the [% %] tags (e.g. lines 3, 5, 7, 8), text-emitting instructions are contained within the [%= %] tags (e.g. [%=wp.a_title%] in line 4), and everything outside these tags is treated as static text.

```
1  \newcommand{\workPackageAndTaskList} {
2    \begin{itemize}
3    [%for (wp in t_wp.all) { %]
4      \item \textbf{ [%=wp.getId()%]  [%=wp.a_title%] }
5      [%for (task in wp.c_task) { %]
6        \subitem Task  [%=task.getId()%]  [%=task.a_title%]
7      [%}%]
8    [%}%]
9    \end{itemize}
10 }
```

Listing 1.2. Excerpt of the proposal model to LaTeX M2T transformation

Having generated a consistent set of commands, we could now import them from the main proposal LaTeX document and use them in arbitrary places. Listing 1.3 illustrates an excerpt of the main proposal document that uses generated LaTeX commands (i.e. *projectDuration, projectName, workPackageAndTaskList, numberOfMilestonesAsWord*).

```
1  \subsubsection{Project Planning - Timeline and Effort
       Distribution}
2  \label{sec:projectPlanning}
3
4  The project duration will be \projectDuration months.
5
6  The \projectName project will be articulated in the following
       work packages:
7
8  \workPackageAndTaskList
9
10 We foresee \textbf{\numberOfMilestonesAsWord milestones} ...
```

Listing 1.3. Excerpt of the main proposal document that uses generated LaTeX commands

Model Validation Our next step was to define validation constraints for project models. We chose to express these constraints using the Epsilon Validation Language [5], for similar reasons to the ones discussed above. In Listing 1.4 we demonstrate two of the defined constraints that are evaluated for all *task*

[5] Naming conventions such as the use of *t_* and *c_* prefixes in Epsilon's XML integration driver are discussed in detail in [4].

elements (*context t_task* in line 1) of the proposal model. The first constraint (*EndAfterStart* in line 3), checks that the start date of a task always precedes its end date (line 4), and produces an appropriate error message if this condition is not met (line 5). Similarly, the second constraint (*constraint WithinTheProject* in line 8) checks that the end date of the task does not extend beyond the project completion date (line 9). To present any identified problems to the contributors of the proposal, unsatisfied constraints produce a new LaTeX command (*generatedWarnings*), which is then imported from the main proposal document.

```
1  context t_task {
2
3    constraint EndAfterStart {
4      check : self.i_end > self.i_start
5      message : "Task " + self.getId() + " ends before it starts"
6    }
7
8    constraint WithinTheProject {
9      check : self.i_end <= t_project.all.first().i_duration
10     message : "Task " + self.getId() + " ends after the end of
           the project"
11   }
12 }
```

Listing 1.4. Validation constraints for PropoGen models

Deployment To enable other contributors to invoke our MDE solution locally, we had to develop and distribute a standalone runnable application. In the interest of simplicity, we developed a self-contained executable Java bundle (JAR) on which users could drag-and-drop their project model to invoke the validation constraints and M2T transformation discussed above. The JAR had to include a complete copy of the EGL/EVL execution engines as well as the actual transformation and validation constraints.

On a side-note, attempting to bundle the Epsilon execution engines into a self-contained JAR file was as useful exercise in itself as collecting and packaging all required dependencies turned out to be quite challenging. As a result of this exercise, Epsilon now provides pre-bundled standalone JAR files[6] that developers can use in their standalone (i.e. non-Eclipse-based) Java or Android applications with minimal effort.

3.3 Unexpected Opportunities

As discussed above, our initial motivation for modelling project proposals was so that we could eventually generate LaTeX content that was tedious and error-prone to maintain manually. However, after developing the LaTeX M2T transformation discussed above, we realised that we could now also produce interesting visualisations for quality assessment purposes from the same model. For example, we developed an additional M2T transformation that generates tree-map

[6] http://www.eclipse.org/epsilon/download/

charts such as the one displayed in Figure 3, which visualises the distribution of effort across different work packages of the project, and Sankey charts, such as the one illustrated in Figure 4 which visualises the contributions of different work packages to the milestones of the project – both of which we have found to be extremely useful for establishing confidence in the balance of the project: one aspect that evaluators tend to study is whether the work/effort/contribution balance is well distributed across partners, themes and work packages (and hence both risk and workload are suitably mitigated and managed).

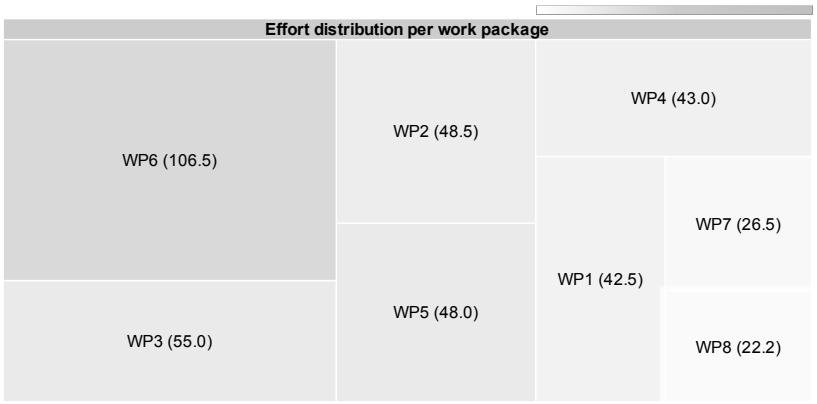

Fig. 3. Tree map visualising the distribution of effort per work package

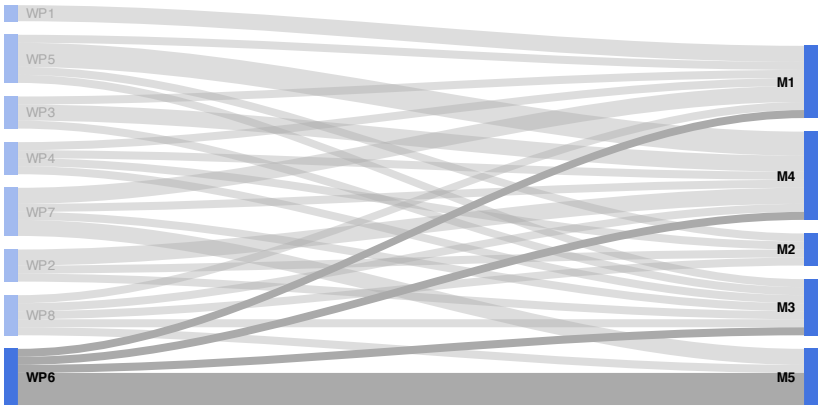

Fig. 4. Sankey diagram visualising how work packages contribute to project milestones

4 Evaluation

In this section we assess the productivity benefits delivered by the XML to LaTeX M2T transformation by analysing data from a recent project proposal that was developed between August 2012 - January 2013. In particular, we measure the number of added, removed and deleted lines of text across consecutive versions of the model and the generated LaTeX commands file, using the *svn diff* and *diffstat* tools as displayed in Listing 1.5. The rationale for doing this is that in the absence of the M2T transformation, we would have needed to perform the same changes to the LaTeX commands file manually.

```
#<file>: the file to diff
#<r1>: older revision number of the <file>
#<r2>: newer revision number of the <file>
svn diff -r <r1>:<r2> <file> | diffstat - m
```

Listing 1.5. Bash command used to calculate the number of changes between consecutive revisions of the XML model and the generated LaTeX commands file

Table 1 presents the obtained measurements. The first column of the table displays the SVN revision numbers for the two files, the second and third columns display the number of changes in the XML model and LaTeX command file with respect to their previous revision in the repository, and the last column displays the difference of these two values. A plotted version of the data displayed in Table 1, appears in Figure 5.

Table 1. Changes in the XML model and generated LaTeX command file by revision

Revision	Changes (XML model)	Changes (generated LaTeX)	Difference
3558	30	38	8
3595	1	46	45
3645	12	29	17
3646	1	3	2
3649	7	13	6
3675	2	5	3
3913	2	21	19
3914	2	9	7
3922	2	8	6
3991	16	31	15
3992	2	6	4
4019	16	75	59
4044	39	43	4
4045	2	5	3
4065	1	2	1
4075	11	39	28
4088	5	15	10
4091	28	42	14
Total	**179**	**430**	**251**

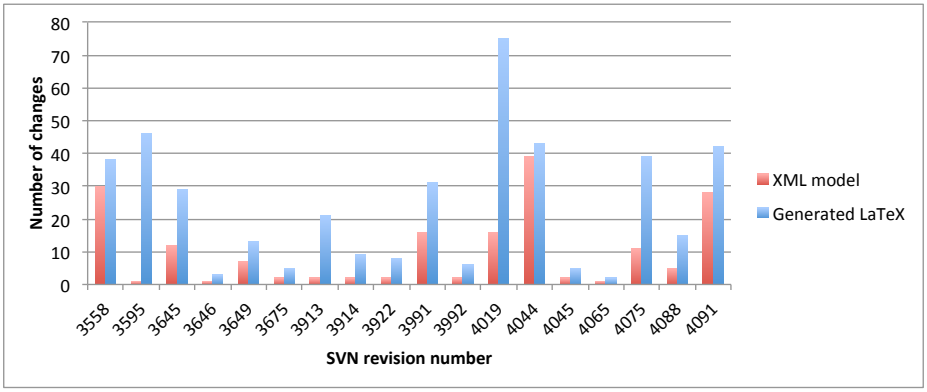

Fig. 5. Plotted data from Table 1

The obtained measurements demonstrate that the M2T transformation delivered a productivity improvement of ∼58% over the lifecycle of the proposal. The latest version of the model for that proposal comprised 256 lines of XML (13,194 bytes) while the generated LaTeX command file comprised 676 lines of dense text (56,694 bytes). Although we do not have hard supporting evidence, it is reasonable to assume that it is also significantly easier and faster to locate and update information in the XML document instead of the LaTeX command file, which has a potential to further amplify the productivity improvement figure obtained above.

4.1 Reflection

Compared to 3-layer metamodelling architectures such as EMF, plain XML is clearly sub-optimal from a technical point of view for capturing interconnected models as it lacks features such as support for cross-references and types. In a non-collaborative environment, we would have most likely used EMF to capture grant proposal models, as this would have also simplified the subsequent model-to-LaTeX transformation.

However, if we were to use EMF in a collaborative environment, we would have needed to implement and distribute standalone language-specific editors (i.e. Eclipse RCP applications) to all partners involved. As RCP applications are platform-specific, we would have needed to export and distribute several permutations of the editor for different operating systems. Moreover, with every change of the metamodel, we would have needed to distribute a new version of the editor application (and most likely deal with the confusion that multiple versions of the same editor can cause).

By choosing to model projects using plain XML, we eliminated the need for developing, maintaining and distributing specialised editors. Despite having some initial concerns about requiring partners to edit XML directly, providing a

comprehensive first version of the XML document appears to have been sufficient even for non-technical partners as we have never – over the last 3 years and 5 grant proposals – received any clarification requests.

Another option we considered early in the design process was to use an off-the-shelf project management tool (e.g. Microsoft Project, ProjectLibre[7]) instead of XML for modelling grant proposals. We decided to use XML instead so that we could have finer control over the structure and organisation of our models.

5 Related Work

There is anecdotal evidence to suggest that several bespoke systems with comparable functionality have been developed and are currently in operation both in academia and industry. This is unsurprising given the size of the domain (over 16,000 proposals were submitted in response to the European Commission's Horizon 2020 calls in April 2014 alone[8]). However, to the best of our knowledge, there is no published work that reports on the organisation, architecture and evaluation of such systems, nor of the specific use cases that such systems aim to support.

In a wider context, several approaches have been proposed for automatically generating system reports, documents and manuals from models in different domains. Hyperdoc [6], is a toolkit that provides support for automated generation of manuals for interactive systems (e.g. VCR players) from state-machine models. Hyperdoc applies graph analysis techniques in order to identify the shortest path between pairs of states and provide efficient instructions to the end-user of the product. In [7], the authors present an approach for generating manuals for families (product lines) of industrial automation systems, using the DOPLER variability modelling tool, DocBook as the target document format, and XSLT for model transformation. In [8], the authors demonstrate how system documents and reports can be generated using a model-based approach from SysML viewpoints and views. In a different domain, in [9], the authors demonstrate how multimedia presentations can be specified at a high level of abstraction using XML, and then compiled using XSLT transformations into concrete artefacts targeting different delivery platforms. XSLT and XML are also used to support processing of highly structured documents with *signing requirements* (e.g., to comply with security policies and governance requirements) in [10], though no explicit metamodel is used in this work.

In [11], the authors provide a systematic review of 34 approaches for generating requirements documents from software engineering models such as UML, user-interface, and goal models and identify a number of best practices including support for 1) bidirectional traceability, 2) structural correspondence between

[7] `http://www.projectlibre.org`

[8] `http://www.sciencebusiness.net/news/76612/Record-numbers-apply-for-Horizon-2020-first-round-funding` (Last accessed: July 1, 2014)

the models and the generated documents, 3) generation of documents in a modifiable format, 4) incremental synchronisation when models change and 5) tailoring the generated document according to its target readership. The approach proposed in this paper is consistent with best practices 2-5 and provides some support for traceability – mainly from the document back to the project model through the generation of LaTeX commands with human-readable identifiers (e.g. `\workPackageOneTitle`).

6 Conclusions

In this paper we have presented how we have applied MDE techniques to automate repetitive and error-prone tasks in the context of the collaborative development of grant proposals. We have demonstrated how grant proposals can be modelled and validated at a high level of abstraction and how model-to-text transformation can then be used to produce correct-by-construction LaTeX macros automating the most tedious and error-prone parts of the process. We regard this application as highly successful and an essential asset for our work on collaborative research projects. Indeed, we have shared this MDE application with partners and colleagues elsewhere, who now use it as part of their proposal development activities.

Acknowledgements

This research was part supported by the EPSRC, through the Large-Scale Complex IT Systems project (EP/F001096/1) and by the EU, through the OSSMETER FP7 STREP project (#318736) and the MONDO FP7 STREP project (#611125).

References

1. Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley, 2008.
2. Dimitrios S. Kolovos, Nicholas Drivalos Matragkas, Horacio Hoyos Rodriguez, and Richard F. Paige. Programmatic muddle management. In *XM@MoDELS*, pages 2–10, 2013.
3. Louis M. Rose, Richard F. Paige, Dimitrios S. Kolovos, Fiona A.C. Polack. The Epsilon Generation Language (EGL). In *Proc. European Conference in Model Driven Architecture (ECMDA)*, 2008.
4. Dimitrios S. Kolovos, Louis M. Rose, Nicholas Matragkas, James Williams, Richard F. Paige. A Lightweight Approach for Managing XML Documents with MDE Languages. In *Proc. 8th European Conference on Modeling Foundations and Applications*, Copenhagen, Denmark, July 2012.
5. Dimitrios S. Kolovos, Richard F. Paige and Fiona A.C. Polack. On the Evolution of OCL for Capturing Structural Constraints in Modelling Languages. In *Proc. Dagstuhl Workshop on Rigorous Methods for Software Construction and Analysis*, 2008.

6. Harold Thimbleby. Combining systems and manuals. In *In Proc. Human-Computer Interaction (HCI) 1993, Volume VIII, BCS*, pages 479–488. University Press, 1993.
7. Rick Rabiser, Wolfgang Heider, Christoph Elsner, Martin Lehofer, Paul Grnbacher, and Christa Schwanninger. A flexible approach for generating product-specific documents in product lines. In Jan Bosch and Jaejoon Lee, editors, *Software Product Lines: Going Beyond*, volume 6287 of *Lecture Notes in Computer Science*, pages 47–61. Springer Berlin Heidelberg, 2010.
8. C. Delp, D. Lam, E. Fosse, and Cin-Young Lee. Model based document and report generation for systems engineering. In *Aerospace Conference, 2013 IEEE*, pages 1–11, March 2013.
9. Lionel Villard, Ccile Roisin, Nabil Layaida, Projet Opra, and Inria Rhne-alpes. An xml-based multimedia document processing model for content adaptation, 2000.
10. Phillip J. Brooke, Richard F. Paige, and Christopher Power. Document-centric xml workflows with fragment digital signatures. *Softw., Pract. Exper.*, 40(8):655–672, 2010.
11. Joaquín Nicolás and Ambrosio Toval. On the generation of requirements specifications from software engineering models: A systematic literature review. *Inf. Softw. Technol.*, 51(9):1291–1307, September 2009.