Science of Computer Programming 134 (2017) 47–60

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Cost-effective evolution of research prototypes into end-user tools: The MACH case study

Harald Störrle

Department of Applied Mathematics and Computer Science, Technical University of Denmark (DTU), Matematiktorvet, 2800 Kongens Lyngby, Denmark

A R T I C L E   I N F O

Article history:
Received 5 March 2014
Received in revised form 28 July 2015
Accepted 29 July 2015
Available online 14 August 2015

Keywords:
Architectural design process
Design decisions
Software tool research prototype
Architectural choice points
UML model analysis and checking

A B S T R A C T

Much of Software Engineering research needs to provide an implementation as proof-of-concept. Often such implementations are created as exploratory prototypes without polished user interfaces, making it difficult to (1) run user studies to validate the tool's contribution, (2) validate the author's claim by fellow scientists, and (3) demonstrate the utility and value of the research contribution to any interested parties. However, turning an exploratory prototype into a "proper" tool for end-users often entails great effort. Heavyweight mainstream frameworks such as Eclipse do not address this issue; their steep learning curves constitute substantial entry barriers to such ecosystems.
In this paper, we present the Model Analyzer/Checker (MACH), a stand-alone tool with a command-line interpreter. MACH integrates a set of research prototypes for analyzing UML models. By choosing a simple command line interpreter rather than (costly) graphical user interface, we achieved the core goal of quickly deploying research results to a broader audience while keeping the required effort to an absolute minimum. We analyze MACH as a case study of how requirements and constraints in an academic environment influence design decisions in software tool development. We argue that our approach while perhaps unconventional, serves its purpose with a remarkable cost-benefit ratio.

1. Motivation

Much of Model Based Software Engineering (MBSE) research needs to provide an implementation as proof-of-concept. For the purposes of research and publication, an exploratory prototype which realizes only the bare minimum in order to test a given novel algorithm suffices. Frequently, little or no effort is spent on issues such as usability, stability, portability, extensibility, and performance. This kind of tool is adequate for demonstrating the concept at hand since only the author will use the tool.

However, if and when the initial goal of proving the concept has been achieved, other goals come into reach.

1. Further scientific questions may arise that can only be answered by human-factors studies, such as observing how users actually use a given tool, and scrutinizing their activities.
2. Fellow scientists (e.g., reviewers) may want to use the tool in order to replicate the results achieved, to compare the new concept/tool to existing ones, or build on top of the previous achievement in other ways.

E-mail address: hsto@dtu.dk.
URL: http://www.compute.dtu.dk/~hsto.

http://dx.doi.org/10.1016/j.scico.2015.07.010
0167-6423/© 2015 Elsevier B.V. All rights reserved.

3. Finally, it may be useful to demonstrate the utility and value of the research contribution to another interested party, e.g., for commercial dissemination, or for getting students and potential collaborators interested in the work.

In our case, our initial interest was to give students access to the tool to support our courses, but clearly, a humble research prototype is not up to any of these challenges.[1]

Unfortunately, turning a research prototype into a "proper" tool implies substantial effort that the original author may not want to spend, since this is usually a pure engineering task with little contribution to the research being conducted. In some rare cases, money may not be a limiting factor, and the development can be outsourced to a commercial software developer. More frequently however, researchers turn such a task into a thesis project for a Bachelor's or Master's student. But that is not always possible: the task may be the wrong size, perhaps no appropriate student is available/willing to do it at a given time, or maybe a chosen candidate does a poor job of the assignment.

All too often, the researcher ends up abandoning the further dissemination of a strand of research for lack of resources. Alternatively, he may find himself implementing a tool, frequently a poor allocation of resources if the resulting tool is intended to attain the degree of polishing seen in commercial or large-scale open-source projects. How can the researcher create a decent tool quickly, and inexpensively? How can the development effort be reduced while still allowing deployment to poorly qualified users such as students? In this paper, we will show our approach to solving this dilemma.

We have archived the current state of the MACH tool on the SHARE platform [19]. It may be used and experimented with online via a remote desktop application using [18]. No installation is required. The archive contains samples and a tool manual.

Outline In the next section, we state our main goals and elaborate them into requirements. We then introduce the main features provided by the Model Analyzer/Checker (MACH), and discuss how those features could be integrated such that they satisfy the given requirements and goals. We evaluate the resulting tool, MACH, against these requirements, and outline two evolutionary paths beyond the work presented in this article. We conclude with a discussion of the results.

2. Goals, constraints, and requirements

The author's research work focuses on advanced operations on UML models that are beyond the scope of existing modeling tools. Over the years, many small exploratory prototypes have been created. While not providing significant practical value individually, their contribution is substantial when combined. Also, they can be valuable in teaching for highlighting model quality issues. In creating the Model Analyzer/Checker tool (MACH), we thus focused on academic stakeholders but neglected industrial users. With this, it is justified to assume that potential MACH users have some understanding of models and the underlying concepts. Also, it is acceptable that the tool does not live up to the levels of quality and usability that are today expected of contemporary professional software. Within academia, we distinguish between *owner* and *user*, where owners are researchers providing a tool and users are students, teachers, or researchers other than the tool owner that want to use the tool for their respective purposes.

Most of these prototypes have been implemented using the PROLOG programming language. The main rationale for choosing PROLOG was to allow rapid prototyping. Long-term usage of the resulting code, usage by third parties, or long-term-evolution of the code base was not considered at the time of creation. This results in three overall goals:

- Combine most or all of the existing prototypes into a single tool;
- Make the major functions available to students and colleagues; but
- Strictly limit the effort in creating the tool to the bare minimum.

These goals are expanded into the requirements shown in Table 1. We then introduce the features offered by MACH and our choice of architectural and technological alternatives in order to satisfy simultaneously all of these requirements.

3. The MACH tool (Model Analyzer/Checker)

In this section we describe the different tools that have been integrated to form MACH. For each tool we detail the benefits realized by integration in MACH, and the difficulties faced in the process of doing so.

As its name suggests, MACH offers advanced analysis and checking procedures on (UML) models. It acts as a proving ground for novel analysis techniques resulting from recent research. The analysis techniques we are interested in could be useful in the context of the Model Driven Engineering (MDE) software development paradigm. MACH currently offers clone detection, version control with semantic aggregation, and size and similarity metrics for UML models. While these features are novel and deserve dissemination in their own right, we present them only as examples. However, first we shall present some convenience functions and the meta meta model that act as architectural glue.

[1] There are several options for addressing these challenges, but by far the most widely used tool platform in MBSE today is Eclipse, in particular the Eclipse Modeling Framework (EMF) [12]. We will discuss alternatives and how they compare with respect to the requirements in detail in Section 4 below.

Table 1
Requirements for MACH, classified by type: **RQA** stands for required quality attributes (aka. non-functional requirements), **C** stands for constraint, and **F** stands for feature.

ID	Type	Description (→Stakeholder)
R1	C	**Existing code base (→Owner)** The research prototypes that shall be integrated into MACH are written in the PROLOG programming language. The existing code base is to be used.
R2	C	**Integration effort (→Owner)** The effort required to create the overall system initially, and to evolve it (e.g., by integrating new features, or combining existing features in novel ways) must be substantially smaller than the effort that was required for initially creating the features (on the order of 1%).
R3	F	**Unifying interface (→User)** There shall be a single, consistent user interface to access all the features bundled in MACH to minimize learning effort.
R4	F	**Convenience functions (→User)** Besides the proper features of interest, auxiliary functions to support practical work will be needed.
R5	RQA	**Portability (→User)** Members of the audiences described above should encounter no substantial trouble installing and running the tool on their personal computers, assuming the use one of one of the widely used operating systems (variants of Windows, Linux, MacOS).
R6	RQA	**Installability (→User)** The two main intended audiences are CS students and researchers; both should be able to install/re-install MACH on a PC within a few minutes.
R7	RQA	**Stability/recovery (→User)** MACH should not terminate unexpectedly. Should such behavior occur anyway, recovery should be quick and convenient (less than 5 s, with no loss of data or state).
R8	C	**Fast bug fixing (→User)** The time needed to analyze problems, create bug fixes, and deploy patches to users shall be less than 1 working day to keep pace with the lecture schedule. Installing a patch must take less than 1 minute.
R9	RQA	**Usability (→User)** CS students and researches should be able to run a sample session of MACH on their own models within minutes and without external help in 9 out of 10 cases.
R10	RQA	**Performance/progress (→User)** For interactive tools like MACH, reaction times below a second are generally recommended. Where this cannot be guaranteed, an indication of progress shall be provided.

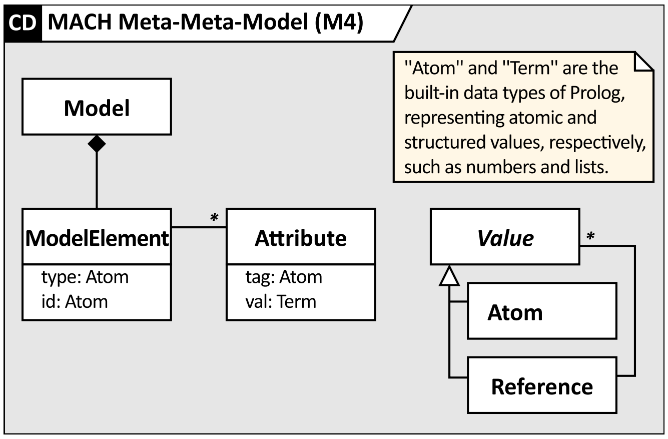
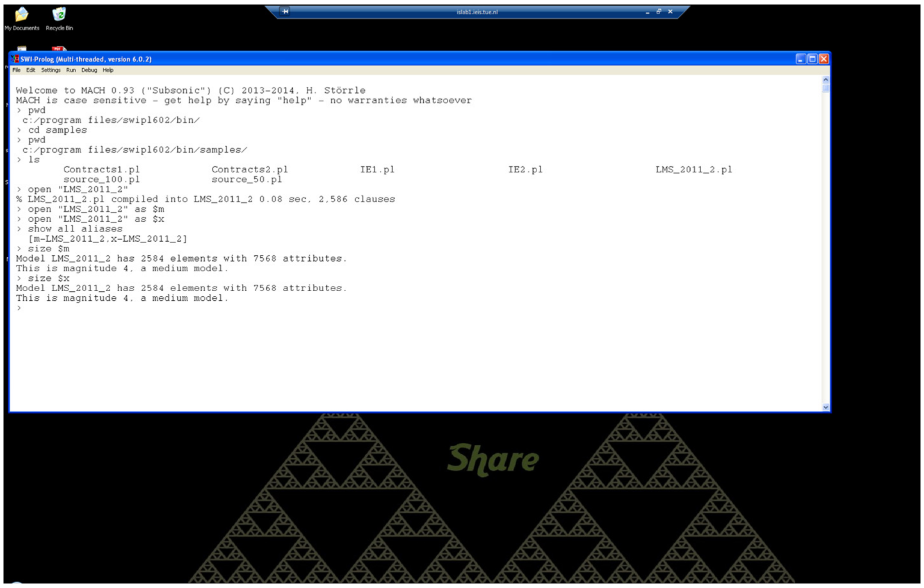

CD MACH Meta-Meta-Model (M4)
Model
ModelElement
type: Atom
id: Atom
*
Attribute
tag: Atom
val: Term
"Atom" and "Term" are the built-in data types of Prolog, representing atomic and structured values, respectively, such as numbers and lists.
Value
*
Atom
Reference

Fig. 1. The Meta Meta Model underlying most MACH tools.

3.1. MACH Meta Meta Model (M4)

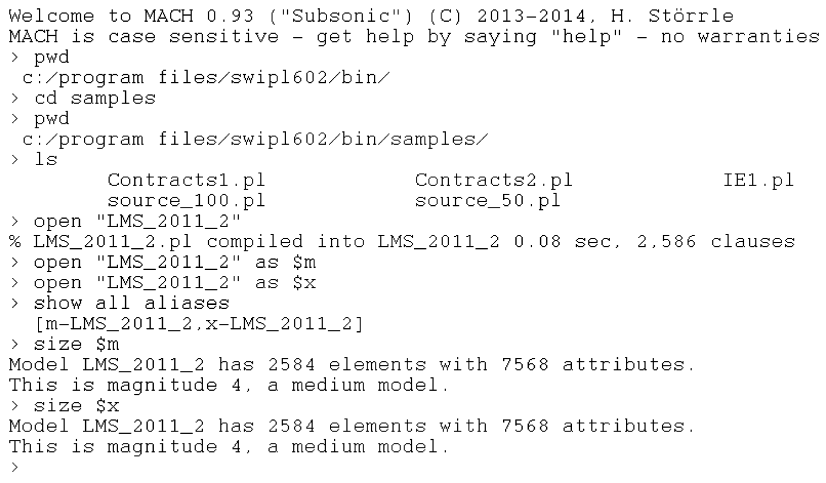
All the tools that are currently integrated into MACH share the common MACH Meta Meta Model (M4) shown in Fig. 1, MACH itself is entirely independent of this structure. This means that one could easily integrate into MACH tools that are governed by different meta models following M4, or even different meta meta models—although often at the expense of not being able to use some of MACH's features. However, M4 is obviously a very generic meta meta model that allows capturing a wide variety of meta models, including those of different versions of UML, BPMN, Matlab/Simulink-models, as well as requirements specification languages.

Depending on the respective tools integrated in MACH, the semantics of the meta model may or may not be exploited. For instance, clone detection relies heavily on UML's model of model structure, while size metrics are independent. Differencing does depend on UML's meta model, but could be made independent relatively easily.


My Documents
Recycle Bin
SWI-Prolog (Multi-threaded, version 6.0.2)
File Edit Settings Run Debug Help
Share

```
Welcome to MACH 0.93 ("Subsonic") (C) 2013-2014, H. Störrle
MACH is case sensitive - get help by saying "help" - no warranties whatsoever
> pwd
 c:/program files/swipl602/bin/
> cd samples
> pwd
 c:/program files/swipl602/bin/samples/
> ls
        Contracts1.pl           Contracts2.pl           IE1.pl                  IE2.pl                  LMS_2011_2.pl
        source_100.pl           source_50.pl
> open "LMS_2011_2"
% LMS_2011_2.pl compiled into LMS_2011_2 0.08 sec, 2,586 clauses
> open "LMS_2011_2" as $m
> open "LMS_2011_2" as $x
> show all aliases
  [m-LMS_2011_2,x-LMS_2011_2]
> size $m
Model LMS_2011_2 has 2584 elements with 7568 attributes.
This is magnitude 4, a medium model.
> size $x
Model LMS_2011_2 has 2584 elements with 7568 attributes.
This is magnitude 4, a medium model.
>
```

Fig. 2. MACH can be explored on the SHARE platform (enlarged details in Fig. 3).



```
Welcome to MACH 0.93 ("Subsonic") (C) 2013-2014, H. Störrle
MACH is case sensitive - get help by saying "help" - no warranties
> pwd
 c:/program files/swipl602/bin/
> cd samples
> pwd
 c:/program files/swipl602/bin/samples/
> ls
        Contracts1.pl           Contracts2.pl           IE1.pl
        source_100.pl           source_50.pl
> open "LMS_2011_2"
% LMS_2011_2.pl compiled into LMS_2011_2 0.08 sec, 2,586 clauses
> open "LMS_2011_2" as $m
> open "LMS_2011_2" as $x
> show all aliases
  [m-LMS_2011_2,x-LMS_2011_2]
> size $m
Model LMS_2011_2 has 2584 elements with 7568 attributes.
This is magnitude 4, a medium model.
> size $x
Model LMS_2011_2 has 2584 elements with 7568 attributes.
This is magnitude 4, a medium model.
>
```

Fig. 3. Basic supportive functions include opening models, assigning an alias, looking up current aliases, and determining the size of a model as numbers of model elements and attributes (enlarged detail from Fig. 2). A command transcript is found in Appendix A.

3.2. Convenience functions

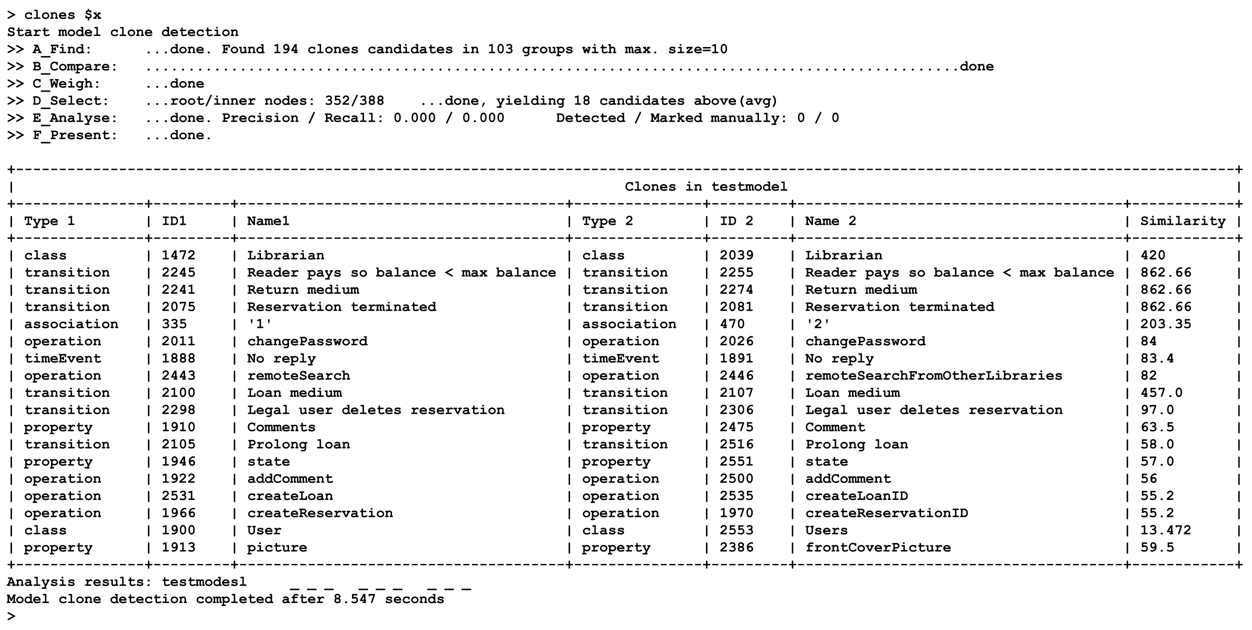
In addition to the features proper as described in the following sections, MACH provides a number of supportive functions that are essential for doing practical work.

The first of these auxiliary components is a package of functions to load models in XMI format, convert them to an internal data structure, and provide aliases in order to reduce typing effort when accessing them. In order to support working with aliases, there are commands for showing and deleting aliases, individually or collectively. This is particularly important, since we frequently use long and very descriptive file names that are easy to mistype or forget.

The second auxiliary component is one that provides facilities to navigate and manage the file system, offering commands much like those well known from UNIX command line interpreters, e.g., `pwd`, `cd`, and so on.

The third component is the textual UI as such. Built on the PROLOG console, it offers a command history, completion of files names and similar convenience features. The fourth and last component is a simple help system that allows users to quickly access information about the available commands and their parameters.

Fig. 2 shows a screen-shot of using MACH on SHARE; Fig. 3 enlarges the terminal window. Appendix A lists the commands issued in the sample session.



```
> clones $x
Start model clone detection
>> A_Find:      ...done. Found 194 clones candidates in 103 groups with max. size=10
>> B_Compare:   ...............................................................................done
>> C_Weigh:     ...done
>> D_Select:    ...root/inner nodes: 352/388    ...done, yielding 18 candidates above(avg)
>> E_Analyse:   ...done. Precision / Recall: 0.000 / 0.000      Detected / Marked manually: 0 / 0
>> F_Present:   ...done.

+--------------------------------------------------------------------------------------------------------------------------------------------------+
|                                                              Clones in testmodel                                                                 |
+--------------+--------+--------------------------------------+--------------+--------+--------------------------------------+------------+
| Type 1       | ID1    | Name1                                | Type 2       | ID 2   | Name 2                               | Similarity |
+--------------+--------+--------------------------------------+--------------+--------+--------------------------------------+------------+
| class        | 1472   | Librarian                            | class        | 2039   | Librarian                            | 420        |
| transition   | 2245   | Reader pays so balance < max balance | transition   | 2255   | Reader pays so balance < max balance | 862.66     |
| transition   | 2241   | Return medium                        | transition   | 2274   | Return medium                        | 862.66     |
| transition   | 2075   | Reservation terminated               | transition   | 2081   | Reservation terminated               | 862.66     |
| association  | 335    | '1'                                  | association  | 470    | '2'                                  | 203.35     |
| operation    | 2011   | changePassword                       | operation    | 2026   | changePassword                       | 84         |
| timeEvent    | 1888   | No reply                             | timeEvent    | 1891   | No reply                             | 83.4       |
| operation    | 2443   | remoteSearch                         | operation    | 2446   | remoteSearchFromOtherLibraries       | 82         |
| transition   | 2100   | Loan medium                          | transition   | 2107   | Loan medium                          | 457.0      |
| transition   | 2298   | Legal user deletes reservation       | transition   | 2306   | Legal user deletes reservation       | 97.0       |
| property     | 1910   | Comments                             | property     | 2475   | Comment                              | 63.5       |
| transition   | 2105   | Prolong loan                         | transition   | 2516   | Prolong loan                         | 58.0       |
| property     | 1946   | state                                | property     | 2551   | state                                | 57.0       |
| operation    | 1922   | addComment                           | operation    | 2500   | addComment                           | 56         |
| operation    | 2531   | createLoan                           | operation    | 2535   | createLoanID                         | 55.2       |
| operation    | 1966   | createReservation                    | operation    | 1970   | createReservationID                  | 55.2       |
| class        | 1900   | User                                 | class        | 2553   | Users                                | 13.472     |
| property     | 1913   | picture                              | property     | 2386   | frontCoverPicture                    | 59.5       |
+--------------+--------+--------------------------------------+--------------+--------+--------------------------------------+------------+
Analysis results: testmodesl
Model clone detection completed after 8.547 seconds
>
```

Fig. 4. Using MACH to detect clones in a model. A command transcript is found in Appendix A.

3.3. *Clone detection*

Code clones are unwanted duplicate snippets of code [8,11], often arising through thoughtless "copy–paste programming". There is compelling evidence that code clones are a substantial impediment to code evolution. Analogously, *model* clones are unwanted duplicate model fragments, and it seems plausible to assume that model clones pose a similar problem to evolving models. Thus, detecting clones is an important task in the quality assurance of models.

In [15,16], model clone detection techniques have been proposed and implemented. The provided tool offers a very large number of parameters, options, and settings that were necessary to experiment with alternative approaches. However, the multitude of controls makes the tool very hard to use without substantial learning effort. We have isolated a configuration that strikes a good compromise for practical purposes, hardwired it in a wrapper around the tool, and plugged it into MACH. Using this function is trivial, and while the results may not be quite as good as when fine-tuning all options, this is the first time such an option has been made easily accessible to non-expert users. In MACH, a modeler simply loads the model to be analyzed, and issues the command `clones "M"` where `M` is the name of a model (see Fig. 4).

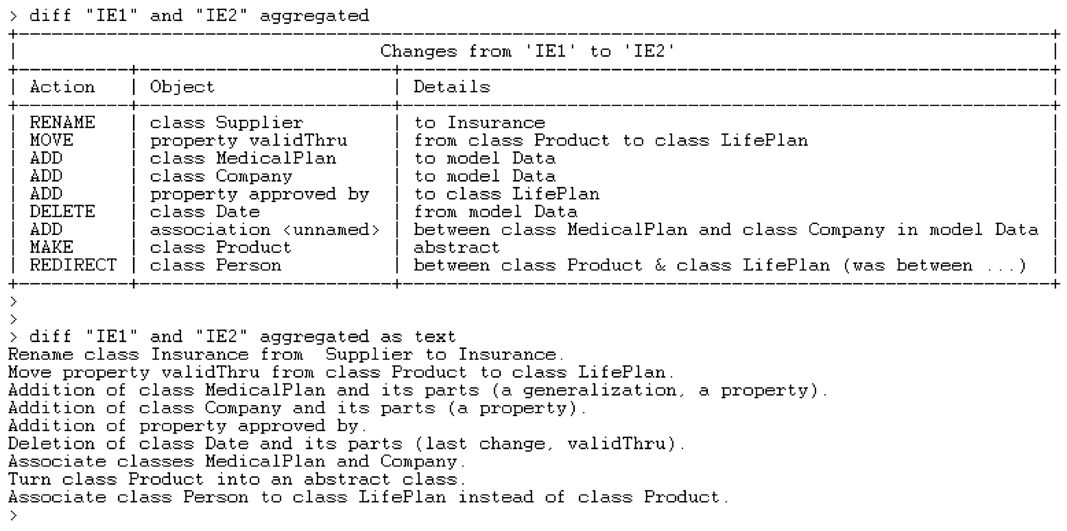
3.4. *Model version control*

There has been a fair amount of research exploring how to best support version control operations for models. One of the topics that has received relatively little attention is the question of how to best present differences to human modelers. In [14], we proposed a novel approach to this problem which outperforms previous solutions. In order to explore other contexts, user studies are needed, requiring a tool good enough to allow such studies.

Probably the largest problem in visualizing the difference between two versions of a model is the inadequate level of detail that model difference engines produce. MACH offers an aggregation function that groups changes into more meaningful units, reducing the number of individual differences drastically. Typically, the reduced difference has less than 30% of the number of the original changes. Currently, the aggregation function is only implemented for class models. See Fig. 5 for an example of the presentation of the difference between two models. An alternative presentation of the model difference as prose is given below the table view.

3.5. *Model size and similarity metrics*

Practical modeling raises the question of how to assess the size and nature of a model. Many model metrics have been proposed, most of them inspired by code metrics [6,7] and restricted to specific model types like class or state chart models; comparing multi-view models as a whole is not supported by such metrics. Consider a long-lasting modeling effort whose progress should be monitored and measured in an objective and automated way. Using the `diff` command presented in the previous section (or similar features implemented in many modeling tools) will likely yield too much detail to allow a comprehensive overview. A metric for the amount of difference from the previous version would be more helpful.



```
> diff "IE1" and "IE2" aggregated
+----------------------------------------------------------------------------------------------+
|                                   Changes from 'IE1' to 'IE2'                                |
+----------+------------------------+----------------------------------------------------------+
| Action   | Object                 | Details                                                  |
+----------+------------------------+----------------------------------------------------------+
| RENAME   | class Supplier         | to Insurance                                             |
| MOVE     | property validThru     | from class Product to class LifePlan                     |
| ADD      | class MedicalPlan      | to model Data                                            |
| ADD      | class Company          | to model Data                                            |
| ADD      | property approved by   | to class LifePlan                                        |
| DELETE   | class Date             | from model Data                                          |
| ADD      | association <unnamed>  | between class MedicalPlan and class Company in model Data |
| MAKE     | class Product          | abstract                                                 |
| REDIRECT | class Person           | between class Product & class LifePlan (was between ...) |
+----------+------------------------+----------------------------------------------------------+
>
>
> diff "IE1" and "IE2" aggregated as text
Rename class Insurance from  Supplier to Insurance.
Move property validThru from class Product to class LifePlan.
Addition of class MedicalPlan and its parts (a generalization, a property).
Addition of class Company and its parts (a property).
Addition of property approved by.
Deletion of class Date and its parts (last change, validThru).
Associate classes MedicalPlan and Company.
Turn class Product into an abstract class.
Associate class Person to class LifePlan instead of class Product.
>
```

Fig. 5. Presenting the difference between two small class models: as a table (top); and as prose (bottom). A command transcript is found in Appendix A.

Such a metric may use the frequency distribution of model element types of different models, easily visualized by histograms (also known as bar charts or spectrogram). This presentation format highlights which concepts have been used frequently and which have been used in-frequently. This allows, for example, to distinguish information models from process models at a glance, without consulting diagrams or other details. In addition, subsequent versions of the same model may be compared with respect to the kind and number of additions and deletions, providing a summary of the extent of changes.

MACH offers several tools to assess the size and nature of a model. For instance, it allows the modeler to present the model as a frequency distribution of the occurrences of meta-classes. Contrasting the distributions of two different models highlights changes in a compact way. If, for instance, an information model is updated by a number of scattered minor changes and the addition of a number of process models to provide a complementary view, this may clearly be seen at a glance using the frequency distribution tool (see Figs. 6 & 7). This application scenario exhibits the limitations of a textual user interface when it comes to graphical visualizations. It also illustrates how MACH mitigates limitations for tables, histograms, and scatter-plots.

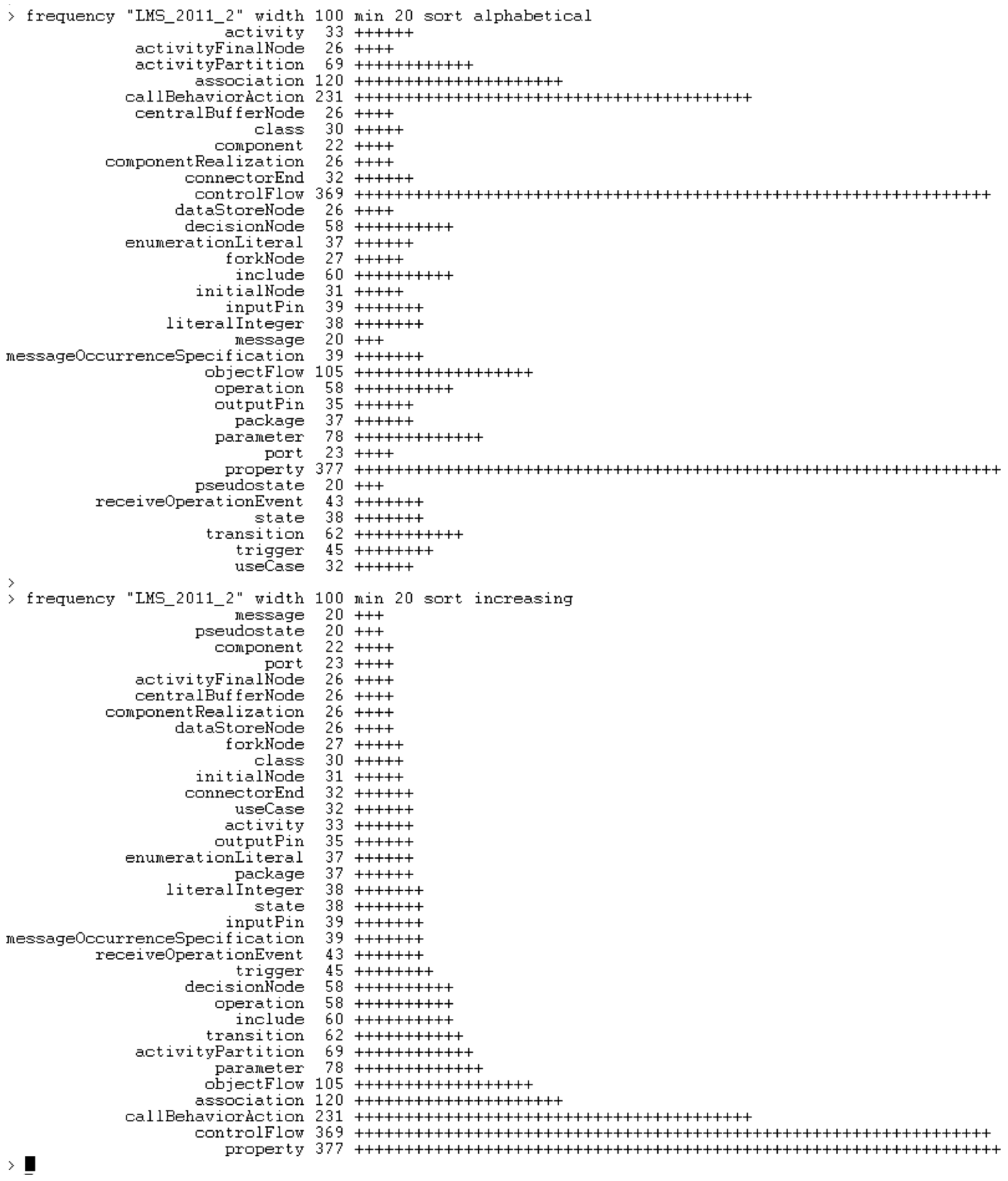
Initial experience suggests, that the concept spectrogram of a model may be rather characteristic. Thus, comparing two models that are *not* from the same lineage may be used as a similarity metric. Consider a second scenario where a teacher needs to assess quickly, whether the models handed in by students as part of their assignment are the result of plagiarism or co-evolution. Manual inspection is obviously too expensive, and it is also relatively easy to mislead a human inspector by superficial changes such as rearranging diagrams, renaming elements, and moving elements around in the containment tree. Computing the identifier overlap will only work for tools that generate identifiers in a deterministic way. Using a cosine similarity based on the meta-class distribution, however, can be used to detect models with similar frequencies of meta class instances.

Finally, consider an organization using MDE as their software development paradigm. Any attempt to control a project by economic parameters crucially relies on a measure of model size being a base component to notions of productivity and project progress. Obviously, the first challenge is to come up with the “right” size metrics of models (see [5,9,13]), justify that it is suitable and indeed better than alternatives, and implement it to demonstrate its capabilities. Clearly, this research agenda requires a tool to compute different model sizes, and easily add/modify implemented metrics. By integrating this component into MACH, we can now easily explore new size metrics and test their effectiveness in class with students.

4. Realization

The components integrated in MACH were all implemented in PROLOG. The problems they address involve symbol manipulation, search, and backtracking, lending themselves quite naturally to the concepts and libraries of PROLOG. While useful for the original research-oriented prototyping, PROLOG is not an ideal choice for building end-user tools. When commercial exploitation is a possibility, a re-implementation with a more mainstream technology might be an option, but in an academic environment, such an approach is unrealistic. Restricted to the original implementations, we are faced with three major design decisions:

1. **Platform:** Should the prototypes be integrated into some kind of platform (and if so which), or combined into a stand-alone tool?



```
> frequency "LMS_2011_2" width 100 min 20 sort alphabetical
                      activity  33 ++++++
             activityFinalNode  26 ++++
             activityPartition  69 ++++++++++++
                   association 120 +++++++++++++++++++++
            callBehaviorAction 231 ++++++++++++++++++++++++++++++++++++++++
             centralBufferNode  26 ++++
                         class  30 +++++
                     component  22 ++++
          componentRealization  26 ++++
                  connectorEnd  32 ++++++
                   controlFlow 369 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                 dataStoreNode  26 ++++
                  decisionNode  58 ++++++++++
            enumerationLiteral  37 ++++++
                      forkNode  27 +++++
                       include  60 ++++++++++
                   initialNode  31 +++++
                      inputPin  39 +++++++
                literalInteger  38 +++++++
                       message  20 +++
messageOccurrenceSpecification  39 +++++++
                    objectFlow 105 ++++++++++++++++++
                     operation  58 ++++++++++
                     outputPin  35 ++++++
                       package  37 ++++++
                     parameter  78 +++++++++++++
                          port  23 ++++
                      property 377 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                   pseudostate  20 +++
         receiveOperationEvent  43 +++++++
                         state  38 +++++++
                    transition  62 +++++++++++
                       trigger  45 ++++++++
                       useCase  32 ++++++
>
> frequency "LMS_2011_2" width 100 min 20 sort increasing
                       message  20 +++
                   pseudostate  20 +++
                     component  22 ++++
                          port  23 ++++
             activityFinalNode  26 ++++
             centralBufferNode  26 ++++
          componentRealization  26 ++++
                 dataStoreNode  26 ++++
                      forkNode  27 +++++
                         class  30 +++++
                   initialNode  31 +++++
                  connectorEnd  32 ++++++
                       useCase  32 ++++++
                      activity  33 ++++++
                     outputPin  35 ++++++
            enumerationLiteral  37 ++++++
                       package  37 ++++++
                literalInteger  38 +++++++
                         state  38 +++++++
                      inputPin  39 +++++++
messageOccurrenceSpecification  39 +++++++
         receiveOperationEvent  43 +++++++
                       trigger  45 ++++++++
                  decisionNode  58 ++++++++++
                     operation  58 ++++++++++
                       include  60 ++++++++++
                    transition  62 +++++++++++
             activityPartition  69 ++++++++++++
                     parameter  78 +++++++++++++
                    objectFlow 105 ++++++++++++++++++
                   association 120 +++++++++++++++++++++
            callBehaviorAction 231 ++++++++++++++++++++++++++++++++++++++++
                   controlFlow 369 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                      property 377 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
> █
```

Fig. 6. Analyzing the occurrences of meta-class instances of a model: ASCII-graphics may be used to visualize the frequency distribution as a histogram: sorting by frequency highlights the extremes and the variation of concept usages in models (top), while sorting by meta-class (bottom) allows direct comparison across different models. A command transcript is found in Appendix A.

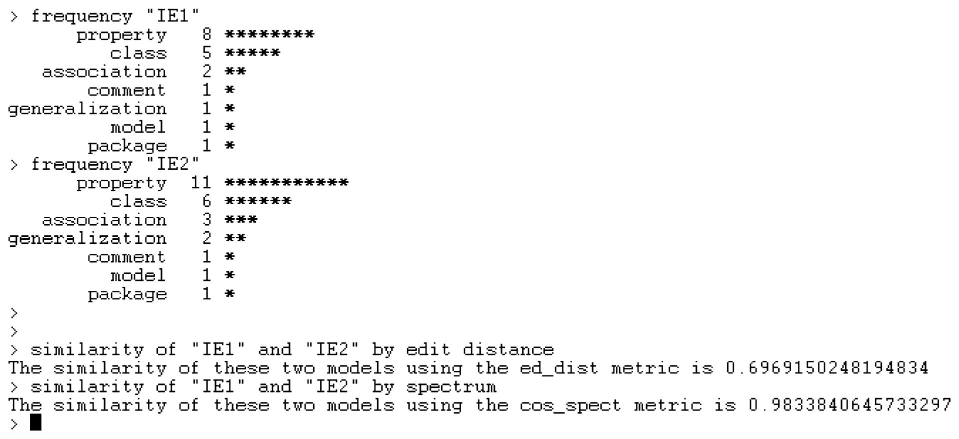
2. **Language:** Which programming language should be used, particularly as the existing code base is in PROLOG?
3. **User Interface:** Should we create a simple command-line user interface or a full-blown graphical UI?

These choice points and our specific decisions are visualized in Fig. 8.

4.1. Platform

Clearly, integrating a tool into a modeling environment provides much easier access to other model-related capabilities, and thus promises a better blending with the modeling process, i.e., higher usability. Additionally, using an existing framework such as Eclipse[2] or a commercial modeling tool with an open API provides a substantial benefit from reusing

[2] Note that we are referring to Eclipse in its role as a framework (i.e., Eclipse RCP), or in its role as a modeling tool (Eclipse plus EMF-related plug-ins), not in its role as an IDE; this discussion is entirely unrelated to IDEs of any kind.



```
> frequency "IE1"
       property   8 ********
          class   5 *****
    association   2 **
        comment   1 *
 generalization   1 *
          model   1 *
        package   1 *
> frequency "IE2"
       property  11 ***********
          class   6 ******
    association   3 ***
 generalization   2 **
        comment   1 *
          model   1 *
        package   1 *
>
>
> similarity of "IE1" and "IE2" by edit distance
The similarity of these two models using the ed_dist metric is 0.6969150248194834
> similarity of "IE1" and "IE2" by spectrum
The similarity of these two models using the cos_spect metric is 0.9833840645733297
> █
```

Fig. 7. Comparing two models by comparing their respective meta-class instance frequency distributions. The histograms immediately highlight the differences. A command transcript is found in Appendix A.

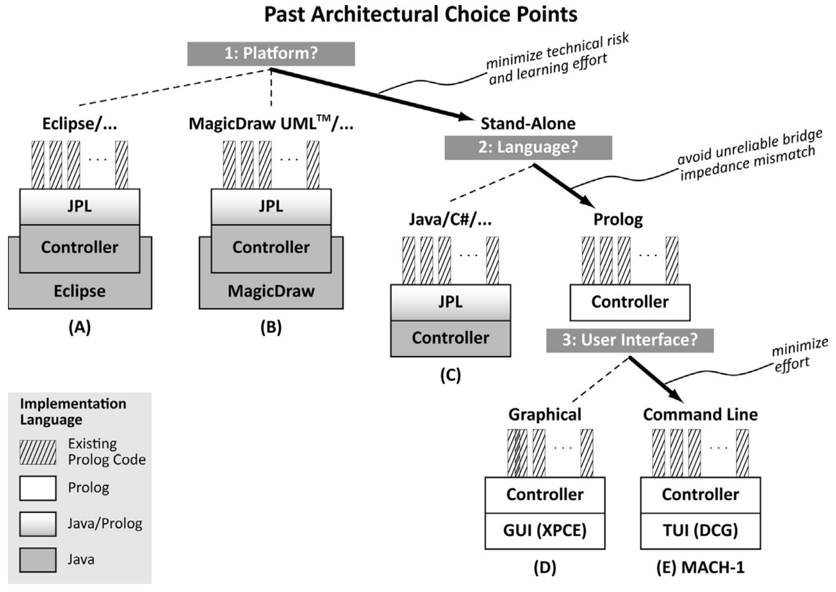


Fig. 8. Architectural alternatives for addressing the requirements of MACH.

the existing code base. The significant drawback is the substantial learning effort required for such an integration. Additionally, using an existing framework incurs the substantial risk that things may not work as smoothly in reality as the documentation promises. Of course, this is a constraint only when this knowledge is not currently present. If a research or development group is already working on a given platform (such as Eclipse) and already has experience in creating tools with this framework, the learning effort may be smaller. Clearly, this is a defining characteristic for *any* framework, to a greater or lesser degree. See [4] for a comparison of Eclipse and PROLOG regarding the learning curve.

Obviously, creating a new modeling environment from scratch would completely negate **R2**. This left us with two realistic candidates to be used as an integration framework. One option was to use the Eclipse Rich Client Platform, a straightforward option because of its openness and widespread use in academia, notably the model-based software development community. However, getting to know any framework can be a significant task, and Eclipse RCP is no exception. In fact, the effort of learning to use Eclipse as a programming framework is so high that such an investment cannot possibly be recouped in the project we consider here.

The other option was to use an existing UML modeling tool as a platform, and implement a plug-in for it. The standard UML modeling tool at our university is MagicDraw UML (MD), which we believe is one of the best commercial UML modeling tools on the market today. Our group had used MD for a long time with very good experiences, it is available on all popular architectures, and it offers a great array of features and plug-ins. There are reasonable licensing and pricing conditions for academic institutions, and there is a so-called team-work server that allows teams to model collaboratively. Crucially, MD also offers an open API, which we had explored previously and found stable and reasonably well-documented. Connecting the prototypes could be achieved using the Java-to-PROLOG library (JPL, see swi-prolog.org). However, an exploratory student project showed at the time that there are inherent technological problems in combining JPL in its current version with MD resulting in instability. Thus incurred an incalculable project risk. Furthermore, the effort required to create the user interface was found to be rather higher than anticipated. Thus, we have ruled out both Eclipse and MD as frameworks for the initial effort, and decided to create a stand-alone tool without integration.

4.2. Language

Choosing Eclipse or MD would have implied using Java, since this is their implementation language. Deciding against these as our implementation platform left us several options. The portability requirement (**R6**) effectively ruled out using the Microsoft technology stack at the given time, e.g., using C# as our implementation language. Java is an obvious choice because of its rich ecosystem and the ability to interface to PROLOG code through the JPL. Also, Java seems to be almost canonical for academic software development. The other straightforward option was to use PROLOG. As it had been used for creating the existing code base there would be no integration effort arising out of a term/object impedance mismatch. There was no reason to consider another language like Python, Ruby, or Tcl/Tk, so the decision was effectively between Java and PROLOG.

Given the known issue with the current version of JPL, we anticipated additional effort from the more complex setting with two programming languages eventually conflicting with requirements **R1** and/or **R2**. Thus we decided to go with PROLOG as our implementation language and specifically SWI-PROLOG (see swi-prolog.org) which we had been using all along.

4.3. User interface

Like the implementation language, the design decisions for the user interface are somewhat constrained by the overall architecture. However, as there are mature freely available GUI-frameworks for PROLOG (e.g., the XPCE library for SWI-PROLOG), we still had the options of using a graphical or command-line interface.

In the light of requirements **R7** (usability), **R8** (fast fixing), and **R3** (single unified user interface), many might consider only a graphical user interface. However, it is surprisingly difficult to create a *good* GUI, and the process involves a large amount of very dull (“boiler plate”) code. A command-line UI (CLI), on the other hand, is much easier to realize. We also expected A CLI to be easier to maintain since the number of files to touch for changes would be very much reduced (down to one, in fact, it turned out). With these considerations in mind, we opted for a textual UI as our first step, with the possibility of upgrading to a graphical UI later if necessary. This is the choice we made for our initial implementation, MACH-1 (see option E in Fig. 8).

5. Evaluation

Triggered by a new teaching assignment, the development and initial deployment of MACH commenced in early 2013. Since then, we have field-tested MACH in 4 graduate and undergraduate classes totaling over 200 students. In each course, we have conducted tool feedback focus groups and anonymous surveys, as well as interviews with teaching assistants to gather information about the usage patterns, problems, and general experience of students using MACH.

*5.1. **R1/R2:** existing code base and integration effort*

The first two requirements were to use the existing code base “as is” with very little integration effort. Clearly, we could reuse the existing code base, and while there were a few changes, those consisted mainly of fixing existing bugs that were discovered in the process of integrating the tools. Some minor additions to the respective APIs were also needed amounting to probably well below 100 touched or added lines of PROLOG code. The MACH system glue code comprises less than 250 lines of PROLOG code. The complete command-line interpreter fits into a single file of less than 200 lines. The underlying code base, on the other hand, comprises over 60,000 LoC (including test cases, general libraries, and some features not yet integrated into MACH). All LoC counts also include comment lines.

The effort expended in creating the research prototypes that formed MACH is impossible to assess accurately, as those prototypes have evolved for over 10 years. However, since the time needed for the integration was in the order of a few days, it is fair to say that the first two requirements are satisfied.

*5.2. **R3:** single unified user interface*

The next requirement demanded a unified interface to access the various integrated prototypes. This has been achieved by a simple command evaluation loop. Implementing this is particularly convenient when using the Definite Clause Grammars (DCGs) built into PROLOG. Changing the command line syntax is also quite simple, contributing to very fast bug fixing times (**R8**).

5.3. ***R4:*** *convenience functions*

Providing convenience functions for file handling consisted of mapping command line syntax to the respective library. There are many small inconsistencies, inadequacies, and gaps in the command handling that result in behavior that differs from the usual command line shells. Still, MACHs usability is good enough to serve as a teaching tool, as documented by the feedback received from students. According to these reports, missing a graphical user interface is considered a much greater problem.

5.4. ***R5:*** *portability*

Students strongly prefer to use their own machines instead of those available in the computing labs. So there is a wide variety of hardware architectures, and operating systems to support. This posed no major problems; MACH runs on recent versions of all major operating systems.

Another portability consideration is the model file format. Even though XMI is a standardized exchange format, existing commercial and academic tools deviate from the standard, which might make them incompatible with MACH. At this point, MACH has only been tested with the XMI produced by MD versions 16.9 to 17.0.3. While this is not an issue in the classroom as students are required to use this tool as part of the course, academics are less likely to have available the specified version of the given tool. Thus, we conclude that requirement **R5** has been satisfied mostly, but not completely.

5.5. ***R6:*** *installability*

Using the students' own machines not only forced the issue of portability, it also underscored the issue of installability. Local settings and varying and/or outdated operating system versions impeded installation in some cases. This led to many students encountering substantial problems installing MACH in the first course in which it was introduced. We overcame this problem by asking students to install a particular version of SWI-PROLOG and then to deploy a saved PROLOG runtime image. So the procedure of re-installing MACH (e.g., after a bug fix) amounted to copying a single file to the proper location.

Since the introduction of this installation procedure, no relevant problems have been reported installing MACH. The very small number of problems we did experience (e.g., those requiring the attention of the teaching assistant, or taking more than 2 minutes for the first installation), all arose from general issues with the respective computer setup (e.g., insufficient rights in the user account to install software on Windows Vista). We therefore conclude, that the installation is simple and without problems, satisfying **R6**.

5.6. ***R7:*** *stability/recovery*

Another concern that we had before deploying MACH was stability. In our experience, a few dozens of students running a tool will expose such issues very quickly. We did not expect such an audience to tolerate incomprehensible error messages, unexpected aborts, and lost work. However, achieving a very high level of reliability in the face of the very diverse platforms we are faced with might be expected to require disproportionate amounts of time and effort.

Instead of optimizing stability, thus, we chose to ensure recoverability. We wrapped the whole of MACH in a global exception handler which issues a generic error message. The "real" error message is hidden and accessible only by specifically asking for it, so that the instructors could actually find out about the underlying problem, while the students would not be confused. This approach worked surprisingly well and allowed the instructors to effectively assist students.

Still, there are some error conditions MACH cannot recover from by itself. In that case the PROLOG command line interpreter appears. Users may then recover manually simply by typing `mach.<CR>`. This process causes very little disruption, since the user's context (e.g., opened files, path settings, screen contents) is maintained across this procedure. So, while MACH's stability is less than perfect, its high recoverability compensates for this shortcoming. We conclude that **R7** is satisfied.

5.7. ***R8:*** *fast bug fixing*

On initial deployment we expected to receive a number of error reports and improvement suggestions. For instance, we were rather uncertain regarding the precise syntax of commands. By implementing the command interpreter by Definite Clause Grammars (DCGs), changing the syntax is indeed very simple, fast, and inexpensive (**R4**). It also allowed adding new (minor) commands or options/parameters, and made the bug fixing cycle very fast (**R3**). Adding a major feature is also quite simple, as long as the feature provides a single command to call the required service. In over two years of continued development, it has proven to be straightforward to gradually wrap features and add them to MACH, as long as the features themselves are reasonably stable. The ease of adding to MACH is largely due to the compact and highly readable code. We conclude that **R3** and **R4** have been addressed successfully.

Table 2
Requirements for MACH and degree to which they have been fulfilled.

ID	Requirement	Satisfaction
R1	Existing code base	Complete
R2	Integration effort	Complete
R3	Unifying interface	Complete
R4	Convenience functions	Partial (features missing)
R5	Portability	Partial (limited data compatibility)
R6	Installability	Complete
R7	Stability/recovery	Complete
R8	Fast bug fixing	Complete
R9	Usability	Partial (sometimes confusing feedback)
R10	Performance/progress	Partial (limited progress)

5.8. ***R9:*** *usability*

Some students also complained about the lack of a (graphical) UI, and there were some that lacked experience with terminal applications and command line tools such as BASH. For some students, issuing a command such as `cd ..` was a challenge, and the concept of a command history appears to be not generally understood among students. We addressed these problems by providing a manual and a cheat-sheet (see [17]), and by giving a brief introduction to the tool, lasting a few minutes, in the respective courses. Since taking these measures, there have been very few complaints or problems regarding the UI, with one exception: many users continue to be puzzled and even irritated by the concept of a text-based UI, reinforcing our earlier observation that our approach is non-standard. After initial hesitation, however, MACH was handled with ease by most students (well in excess of 90%), including ones that major in non-CS programs. We therefore expect, that most users can use MACH without great difficulty. While MACH will probably never be very popular for its user interface, it does serve its purpose. We thus conclude that **R9** has been satisfied.

5.9. ***R10:*** *performance/progress*

While models in typical teaching settings are relatively small, and performance is not likely to be an issue, models may be larger in research settings. For models with several thousands of elements, some features (e.g., clone detection) may have response times in excess of three seconds. While the performance of the integrated features is outside the scope of MACH, MACH should provide sensible feedback to the user, e.g., some kind of progress indication. We deferred this task to the integrated components, and simply passed their output and progress indication to the command line. The overhead incurred from the integration of features is negligible.

Table 2 revisits the requirements and summarizes the degree to which they have been satisfied. Observe that some of the limitations are rooted in the underlying research prototypes rather than the integration effort.

6. Future evolution and limitations

Since the PROLOG language lends itself very naturally to problems involving symbol manipulation, search, and backtracking, it was a good choice as an implementation language used to develop the original research prototypes. However, PROLOG is not a mainstream language making it hard to find developers with suitable competencies. Also, most existing tools and frameworks are based on more conventional languages like Java and C#, and there is usually no easy way to connect the two technologies. This makes Prolog difficult to integrate smoothly into existing tools and frameworks. It is also difficult to attract skilled developers who want to use PROLOG. Finding students well versed in Java or C# is much easier. Additionally, installing MACH requires installing SWI-Prolog first while Java is usually installed already. Finally, providing MACH as a stand-alone tool creates an integration gap between the model editor and the model analysis tool negatively impacting the overall workflow. The question therefore arises about how to evolve MACH in the future in order to overcome these problems.

Our first approach to this challenge was a by-product of a student project [1]. It provided a plug-in to MagicDraw that was capable of running MACH-1 embedded in a plug-in (see option F in Fig. 9), thus yielding a version of MACH fully integrated into MagicDraw at no extra cost. However, that solution suffers from instabilities caused by the underlying Java to Prolog Bridge JPL, which making it unsuitable for this use.

In a subsequent step, we have experimented with converting MACH into a web-based tool (option G in Fig. 9). This resulted in the Hypersonic web-service that offers one of the services of MACH (model clone detection) via a web page.[3] Clearly, this kind of interface requires no installation or unusual infrastructure and is thus useful for reaching a very large

[3] Hypersonic is available via the MACH homepage http://www2.compute.dtu.dk/~hsto/tools/mach.html or directly at http://www2.compute.dtu.dk/~rvac/hypersonic. See [2,3] for references.

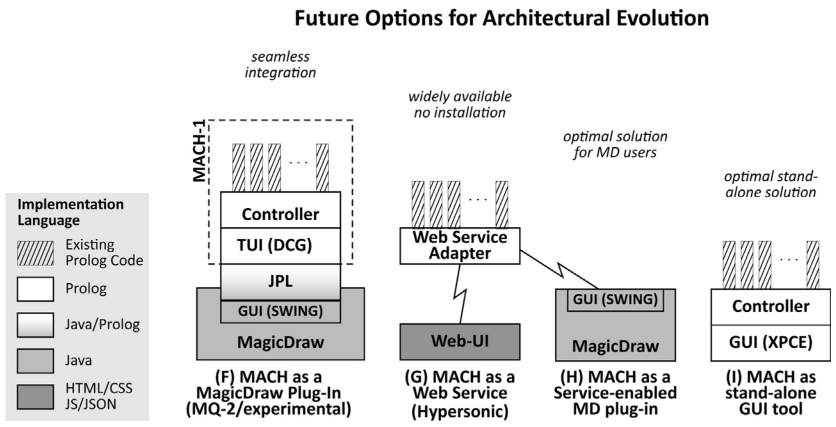

Future Options for Architectural Evolution
Implementation Language
Existing Prolog Code
Prolog
Java/Prolog
Java
HTML/CSS JS/JSON
seamless integration
MACH-1
Controller
TUI (DCG)
JPL
GUI (SWING)
MagicDraw
(F) MACH as a MagicDraw Plug-In (MQ-2/experimental)
widely available no installation
Web Service Adapter
Web-UI
(G) MACH as a Web Service (Hypersonic)
optimal solution for MD users
GUI (SWING)
MagicDraw
(H) MACH as a Service-enabled MD plug-in
optimal stand-alone solution
Controller
GUI (XPCE)
(I) MACH as stand-alone GUI tool

Fig. 9. Architectural alternatives for the future evolution of MACH: options F and G are currently implemented experimentally with reduced feature sets.

audience. The implementation effort for this solution was very modest indeed, suggesting that a more comprehensive implementation of MACH with this technology is not just possible, but actually more cost-effective than any of the alternatives. Additionally, this approach offers a smooth evolution path for experimental tools: these kinds of web services can be added rather easily as well as incrementally.

While web services still do not provide the seamless interfaces provided by tight integrations into a modeling tool like MagicDraw or a framework like Eclipse, a convenient interface is easily and cheaply provided to a very wide audience. Providing a tool as a web service allows for updates and enhancements, but does not offer the archival qualities offered available from services like SHARE, unless the whole web-server were to be deployed there. We are currently investigating another approach where a front end to the existing web-services is created as a plug-in to MagicDraw (option H in Fig. 9). This would combine the seamless integration of a native feature inside the modeling environment with the flexibility and generality of a web-service based solution making this the best option for users of MagicDraw. Of course, users of other UML tools would have no added value from this solution since a different plug-in is needed for every other tool.

The best option for a generic stand-alone tool would be to replace the command-line interface with a graphical user interface based on XPCE [20], the SWI-PROLOG native GUI library (option I in Fig. 9). Obviously, in options G, H, and I adapters are still needed to bridge any file format incompatibilities between the XMI dialects used by MagicDraw and other tools.

7. Conclusions

In this paper we presented the MACH tool and outlined the forces that influenced its design process. MACH is implemented in PROLOG and provides a command-line user interface, both of which are relatively exotic choices, today. MACH can be obtained online at www.compute.dtu.dk/~hsto. There is an online demonstration of MACH 0.93 available via SHARE at [18]. Usage experience so far suggests that our approach was successful in satisfying the goals associated with it.

In previous sections, we have outlined the general setting of this project, how it gives rise to the requirements, and how (and to what degree) we have addressed them in MACH:

- We succeeded in using the existing code base without modifications (**R1**), integrating it with a unifying interface (**R3**), and equipping it with a set of convenience functions (**R4**) producing functionalities which, while not comparable to those available in professional tools and shells, are currently sufficient for the given applications. All of this was achieved with very little effort (**R2**). Thus, the main objectives were achieved.
- Field testing MACH raised only a few portability issues (**R5**). The original installation procedure was inadequate, but could be replaced quickly and cheaply (**R6**). MACH runs generally very reliably, and recovering from exceptional behavior is very fast and simple (**R7**). Fixing minor bugs and redeployment went smoothly (**R8**).
- The biggest unknown of our exploration was whether the idea of providing an (inexpensive) command-line interface would be good enough to serve the intended audience and purpose. The combined feedback from over 200 students using MACH demonstrates that despite the usual idiosyncrasies, and a fair number of oddities and shortcomings of the user interface, it can be used effectively for the intended purposes in its current state. Thus, requirements **R9** and **R10** have also been satisfied.

We have succeeded in making advanced functionality available to students and other interested parties at very little cost. The experience of creating MACH has clearly shown that there are architectural alternatives to Eclipse RCP and similar frameworks, and that a command-line UI can serve its purpose just as well as a graphical UI—at lower cost. Creating a light-weight tool with a textual interface can be achieved with very little effort. At a time when developers, academics, and students alike are used to highly polished IDEs, and have come to expect high levels of usability in tools, our approach may appear to be unconventional, even old-fashioned or unsophisticated. It does, however, effectively serve its purpose, in a pragmatic way and with a unique cost-benefit ratio.

We believe that this is a path other researchers faced with similar challenges might be able to follow. We hope that our experience will help others make their research prototypes available to a larger audience with reasonable effort, for the benefit of the entire scientific community.

Appendix A. Sample session transcript

The screen-shots presented in this paper have been created with the archival installation of MACH available on the SHARE platform. The following is a transcript of the user commands issued to obtain these screen-shots. Observe that parameters starting with capital letters or containing numbers or special characters (white-spaces, underscores, dashes etc.) must be enclosed in double quotes.

```
# Commands issued in Figure 1 & 2
> pwd
> cd "samples"
> pwd
> ls
> open "LMS_2011_2"
> open "LMS_2011_2" as $x
> open "LMS_2011_2" as $m
> show all aliases
> size $x

# Command issued in Figure 3
> clones $x

# Commands issued in Figure 4
> diff "IE1" and "IE2" aggregated
> diff "IE1" and "IE2" aggregated as text

# Commands issued in Figure 5
> frequency $x width 100 min 20 sort alphabetical
> frequency $x width 100 min 20 sort increasing

# Commands issued in Figure 6
> frequency "IE1"
> frequency "IE2"
> similarity of "IE1" and "IE2" by edit distance
> similarity of "IE1" and "IE2" by spectrum
```

References

[1] Vlad Acretoaie, Harald Störrle, MQ-2: a tool for Prolog-based model querying, in: Proc. Eur. Conf. Modelling Foundations and Applications, ECMFA, in: LNCS, vol. 7349, Springer Verlag, 2012, pp. 328–331.

[2] Vlad Acretoaie, Harald Störrle, Hypersonic – model analysis as a service, in: Stefan Sauer, Manuel Wimmer, Marcela Genero, Shaz Qadeer (Eds.), Joint Proc. MODELS 2014 Poster Session and ACM Student Research Competition, in: CEUR, vol. 1258, 2014, pp. 1–5, available online at http://ceur-ws.org/Vol-1258.

[3] Vlad Acretoaie, Harald Störrle, Hypersonic: model analysis and checking in the cloud, in: Dimitris Kolovos, Davide DiRuscio, Nicholas Matragkas, Juan De Lara, Istvan Rath, Massimo Tisi (Eds.), Proc. Ws. BIG MDE, 2014.

[4] Don Batory, Eric Latimer, Maider Azanza, Teaching model driven engineering from a relational database perspective, in: Ana Moreira, Bernhard Schätz, Jeff Gray, Antonio Vallecillo, Peter Clarke (Eds.), 16th Intl. Conf. Model Driven Engineering Languages and Systems, MoDELS'13, in: LNCS, vol. 8107, Springer Verlag, 2013, pp. 121–137.

[5] M.R. Chaudron, B. Cheng, C. Lange, J. McQuillan, A. Nugroho, A. Neczwid, F. Weil (Eds.), Proc. 2nd Ws. Model Size Metrics, MSM'07, 2007, available at http://www.win.tue.nl/~clange/MSM2007/MSM2007.pdf.

[6] Marcela Genero, Mario Piattini, Coral Calero, A survey of metrics for UML class diagrams, J. Object Technol. 4 (9) (Nov/Dec 2005) 55–92.

[7] Marcela Genero, Mario Piattini, Coral Calero, Metrics for Software Conceptual Models, Imperial College Press, London, 2005.

[8] Rainer Koschke, Survey of research on software clones, in: Andrew Walenstein, Rainer Koschke, Ettore Merlo (Eds.), Duplication, Redundancy, and Similarity in Software, Dagstuhl Seminar 06301, Intl. Conf. and Research Center for Computer Science, Dagstuhl Castle, 2006.

[9] Andrij Neczwid, Frank Weil (Eds.), Proc. 1st Ws. Model Size Metrics, MSM'06, 2006, see summary in [10].

[10] Oscar Nierstrasz, Jon Whittle, David Harel, Gianna Reggio (Eds.), Proc. 9th Intl. Conf. Model Driven Engineering Languages and Systems, MoDELS'09, LNCS, vol. 4199, Springer Verlag, 2006.

[11] Chanchal K. Roy, James R. Cordy, A survey on software clone detection, Technical report TR 541, Queen's University, School of Computing, 2007.

[12] Dave Steinberg, Frank Budinsky, Ed Merks, Marcelo Paternostro, EMF: Eclipse Modeling Framework, Pearson Education, 2008.
[13] Harald Störrle, Large scale modeling efforts: a survey on challenges and best practices, in: Wilhelm Hasselbring (Ed.), Proc. IASTED Intl. Conf. Software Engineering, Acta Press, 2007, pp. 382–389.
[14] Harald Störrle, Making sense to modelers – presenting UML class model differences in prose, in: Joaquim Filipe, Rui César das Neves, Slimane Hammoudi, Lui Ferreira Pires (Eds.), Proc. 1st Intl. Conf. Model-Driven Engineering and Software Development, SCITEPRESS, 2013, pp. 39–48.
[15] Harald Störrle, Towards clone detection in UML domain models, J. Softw. Syst. Model. 12 (2) (2013) 307–329.
[16] Harald Störrle, Effective and efficient model clone detection, in: Rocco De Nicola, Rolf Hennicker (Eds.), Software, Services and Systems, in: LNCS, Springer Verlag, 2014.
[17] Harald Störrle, MACH 0.92 (subsonic), DTU technical report 2014-02, Technical University of Denmark (DTU), 2014.
[18] Harald Störrle, Online demo: Mach 0.93, http://is.ieis.tue.nl/staff/pvgorp/share/?page=ConfigureNewSession&vdi=XP-TUe_XP_SWIPL.vdi, 2014.
[19] Pieter Van Gorp, Steffen Mazanek, SHARE: a web portal for creating and sharing executable research papers, Proc. Comput. Sci. 4 (2011) 589–597.
[20] Jan Wielemaker, Anjo Anjewierden, Programming in XPCE/Prolog, Technical report, SWI, University of Amsterdam, 1992/2005.