DesignSpace – An Infrastructure for Multi-User/Multi-Tool Engineering

Andreas Demuth
Institute for Software Systems Engineering
Johannes Kepler University
Linz, Austria
andreas.demuth@jku.at

Markus Riedl-Ehrenleitner
Institute for Software Systems Engineering
Johannes Kepler University
Linz, Austria
markus.riedl@jku.at

Alexander Nöhrer
Institute for Software Systems Engineering
Johannes Kepler University
Linz, Austria
alexander.noehrer@jku.at

Peter Hehenberger
Institute of Mechatronic Design and Production
Johannes Kepler University
Linz, Austria
peter.hehenberger@jku.at

Klaus Zeman
Institute of Mechatronic Design and Production
Johannes Kepler University
Linz, Austria
klaus.zeman@jku.at

Alexander Egyed
Institute for Software Systems Engineering
Johannes Kepler University
Linz, Austria
alexander.egyed@jku.at

ABSTRACT

The engineering and maintenance of large (software) systems is an inherently collaborative process that involves diverse engineering teams, heterogeneous development artifacts, and different engineering tools. While teams have to collaborate continuously and their artifacts are often related, the tools they use are nearly always independent, single-user applications. These tools range from programming to modeling tools and cover a wide range of engineering disciplines. However, relations among the artifacts across these tools often remain undocumented and are handled in an ad-hoc manner. Keeping these artifacts in sync continues to be a key engineering challenge. In this paper, we present our vision of the *DesignSpace*, a novel engineering infrastructure for integrating diverse development artifacts and their relations. The DesignSpace supports distributed collaboration, a wide range of tools and development, maintenance , and evolution services including incremental consistency checking and transformation.

1. INTRODUCTION

The engineering and maintenance of software systems often requires expertise from various disciplines and involves diverse tasks that range from requirements engineering to HW/SW modeling to testing and deployment. Most of these tasks are supported by highly specialized engineering tools of excellent quality that let engineers produce many kinds of development artifacts quite efficiently. However, it is important to note that most of these engineering tools are first and foremost single-user applications, focusing on the tasks of individual engineers and specific kinds of artifacts (e.g., Eclipse for Java code, ProEngineer for 3D-CAD drawings, or Matlab for modeling and simulation). While these engineering tools are separate, it is clear that the artifacts created with them belong together. For example, the software engineer must understand the mechanics of a system and must conform to its behavior even though this behavior is decided by a mechanical engineer. The software and hardware thus needs to be in sync and failure to identify discrepancies will cause the system to fail. We refer to such discrepancies as inconsistencies, which are particularly hard to catch if they involve artifacts that are modeled in different engineering tools, possibly involving different engineers and different engineering domains.

This disconnect between the standalone nature of engineering tools and the inherently collaborative nature of engineering implies that engineers need to divert attention from the actual engineering task to manually ensure consistency across development artifacts and to manually propagate changes among them. Doing so is recognized to be error prone as different artifacts might be closely related in terms of their purpose and semantics, but they are likely to have different metamodels or languages, and they might even have different file formats that require different tools for editing. Consequently, two engineers who are responsible for two different artifacts might use similar concepts and ideas, but in such different languages and tools that neither involved engineer can perform the necessary changes in a consistent manner. For example, a mechanical engineer is able to describe and simulate a dynamic behavior in Matlab but does not have the programming skills to reflect this behavior in the software system. In reverse, a software engineer can implement the robot controller that conforms with the expected behavior but may not be able to understand where this behavior comes from or even why it is this way. Engineering is about sharing knowledge and it is about collaboration. Understanding who needs to communicate with whom is a hard problem. But so is remembering this communication for later change. Where exactly is this robot behavior implemented in the software controller? What do we need to change if this robotic behavior changes? Which engineer is best suited to perform this change. And, what subsequent changes might the change to the software controller imply? Without understanding the relations among engineering artifacts involved—e.g., the Matlab computational model and the software—the engineers performing the changes might not be able to collaborate effectively. Even worse, the lack of such re-

SAC'15 April 13-17, 2015, Salamanca, Spain.

http://dx.doi.org/10.1145/2695664.2695697

lations might cause engineers to be unaware of the need for some collaboration and propagating the change only partially.

Considering frequent artifact changes, the task of keeping artifacts consistent and propagating changes completely and correctly is perhaps the foremost engineering challenge today. The process is mostly ad-hoc and while standards may require relations to be captured without automated support they provide little protection against development artifacts drifting apart. This problem is well recognized and tied to exploding engineering costs [1] and engineering failures [2].

To address the current lack of support for efficient collaboration among engineers across artifacts and tools, this new idea paper outlines our vision of an engineering infrastructure—the *DesignSpace*—that supports the integration of arbitrary engineering artifacts and provides engineering services for handling artifact evolution and engineer collaboration in an effective and efficient manner.

2. PROBLEM ILLUSTRATED

To illustrate the issue of unmanaged artifact relations, we use an excerpt of a robot system. The role of a robot is to pick up objects with a gripper hand and move them to a new position. Figure 1 depicts a small set of engineering tools and a few artifacts that are involved in the project. Each tool handles a specific kind of artifact. For example, a mechanical engineer may use a spreadsheet to compute the length of a robot arm based on the distance the robot ought be able to reach (perhaps taken from the requirements specification) and the min/max angles the arm segments are able to rotate (perhaps stemming from the physical constraints of the arm joints modeled elsewhere).

None of these engineer tools require another tool to function, they are separate. However, not separate are the artifacts that engineers create using these tools. There is typically a significant amount of knowledge shared among engineers. Unfortunately, this knowledge sharing is typically done in an ad-hoc, implicit manner: once knowledge is created, it is shared by engineers through informal communication and used to produce development artifacts. Thus, development knowledge often remains *tacit* and it is only documented and shared implicitly in form of its manifestation (e.g., specific design decisions) in different artifacts. Figure 1 also depicts the tacit knowledge sharing in the project. For example, we see that the arm length computed in the spreadsheet was then propagated to various places. Specifically, see that the CAD drawing requires the correct arm length to depict the robot. Moreover, the software controller for the robot requires the arm length to compute rotation angles needed to correctly position the arm's servo motors for grabbing objects.

For this work, it is irrelevant how the knowledge about the "arm length" was shared (e.g., oral communication through a phone call, or written communication using email). What is relevant is that different development artifacts in different tools have shared knowledge and inter-dependencies. The tools and their respective artifacts remain independent because each tool involved is perfectly able to model its respective artifacts without knowledge about the other tools. However, all three tools fail to retain the knowledge about artifact dependencies. For instance, consider a change of the maximum reachable "distance" in the requirements specification. Except for the requirements specification itself, which artifacts are affected and how should they be evolved? Unfortunately, there is little to no support for engineers to answer this question. Typically, engineers need to manually identify the exact spreadsheet that requires updating (there can be a large number of such sheets). Or, engineers need to manually change the code because the robot's grasping "distance" is manifested there as the "Robot.maxDistance". And this is not enough. A change in the robot's grasping distance may cause subsequent changes such as possibly a modified arm length. What is affected by that? This is known as a ripple effect that makes change propagation difficult because the failure to identify all artifacts that are directly or indirectly affected by a change implies the failure to propagate a change correctly and/or completely. Note that this example focused on a trivial, albeit common form of knowledge sharing: the replication of a value. However, knowledge sharing can be far less trivial by involving multiple artifacts and complex constraints.

3. DESIGNSPACE VISION

This paper discussed our vision of a collaborative environment, called the DesignSpace, that lets engineers share knowledge and record inter-dependencies—even and especially if the shared knowledge or inter-dependencies span across engineering tool boundaries. Engineers may continue to work in private (i.e., to explore ideas that are not yet ready to be shared with others) but they may also work together in protected groups or even in an publicly visible manner. Engineers may continue to use their favorite engineering tools as-is and to follow their preferred style of collaboration (e.g., an SVN-like checkout-change-commit workflow with private working areas) but we want engineers to also be able to provide knowledge that is currently not (elegantly) expressible in their tools. This includes the ability to define domain knowledge (e.g., domain models) or to link dependent artifacts from different tools and even domains together. Making those artifact relations explicit not only documents *traceability* between artifacts, but it also enables automatic support for checking *consistency* and reacting to changes during the *evolution* of the system. For any artifact change, regardless of whether it is visible publicly or performed in a private working area, information should be provided to engineers instantly that answers the following three questions: 1) Which artifacts (i.e., part of the system) are involved in inconsistencies or are affected by a change? 2) Which engineers are responsible for the affected artifacts? 3) Which tools can be used to repair these artifacts and what repair options are there? Furthermore we envision that the whole evolutionary history will be kept at a quite fine level of granularity to allow a time-faithful reconstruction of how a design came about.

4. APPROACH OUTLINE

The *DesignSpace* is a cloud-based artifact integration and service platform that enables effective and efficient collaboration by explicitly sharing knowledge that is otherwise hidden inside engineering tools. It makes publicly available the development artifacts and allows arbitrary inter-dependencies to be established—hence augmenting the knowledge that is already available in development tools. Moreover, the DesignSpace provides services that enable engineers to work with diverse artifacts, to handle inconsistencies, to perform transformations, and to collaborate efficiently.

4.1 Data Services

The DesignSpace is "aware" of engineers working synchronously and asynchronously, the engineering tools they are using, and even the artifacts they are working on. The DesignSpace thus provides "virtual" access to artifacts in engineering tools. This key feature allows engineers to use the DesignSpace to define inter-dependencies among artifacts—even if the artifacts are modeled in different engineering tools.

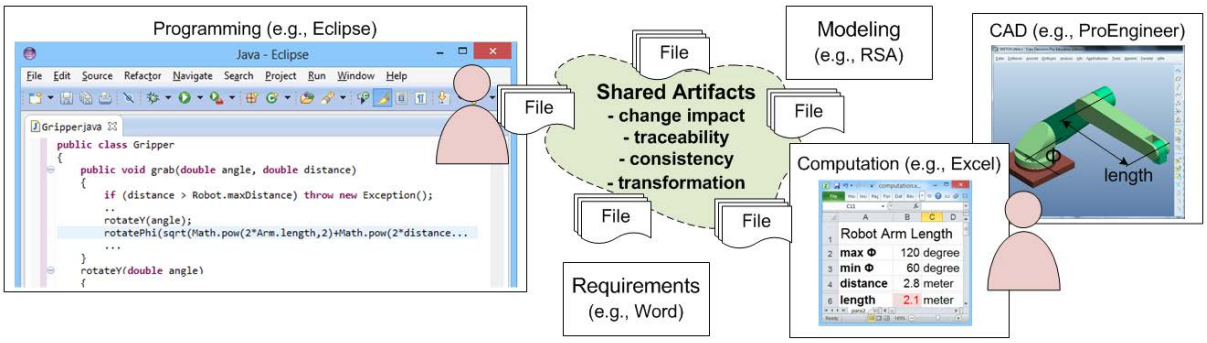

Figure 1: Illustration of some Artifacts and Tools used during the Engineering of a Robot System.

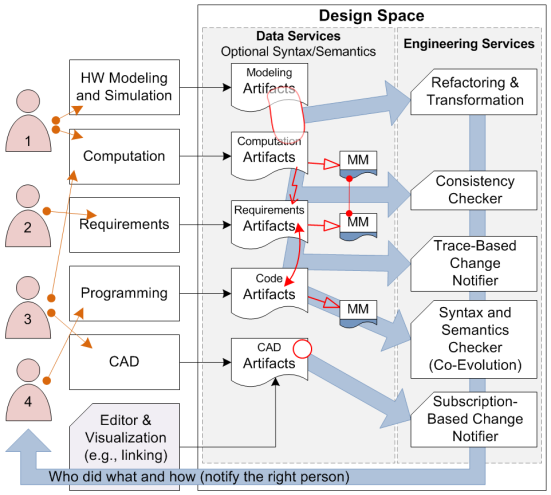

Figure 2: DesignSpace Data and Engineering Services.

To support both synchronous and asynchronous collaboration among engineers, the DesignSpace does not actually access artifacts in engineering tools but rather the engineering tools are made to instantly propagate artifact changes to the DesignSpace. The DesignSpace is thus a mirror of artifacts found in its corresponding engineering tool. To keep artifacts in the engineering tools in sync with the artifacts in the DesignSpace, artifact- or tool-specific adapters are necessary. These need to be build separately for each engineering tool integrated with the DesignSpace. We support two manners of artifact sharing between engineering tools and the DesignSpace: *i)* complete artifact sharing, and *ii)* selected artifact sharing limited to artifacts that have been specifically marked by engineers. We found the latter to be of particular interest to the engineering community because for many complex calculations or simulations (e.g., Matlab) only some key parameters need to be shared with others or linked (e.g., their input and output but not necessarily the computations within). For example, not the entire CAD drawing in Figure 1 may be of interest to others but key parameters only such as the length of an arm. The tool adapters ideally sync artifact changes with the DesignSpace instantly. However, it is also possible to queue the changes if the engineering tool is used offline.

To represent engineering tool artifacts, the DesignSpace provides a uniform data structure. Development artifacts are then encoded as *nodes* with *properties* and *references* to other nodes—analogous to ontologies or other metamodeling languages, such as EMF/ECORE (though we will see later that there are in fact strong differences). For example, once a UML model and code have been synced to the DesignSpace, linking an UML operation in a class diagram to its corresponding Java method can be done simply by adding an edge between the two nodes that reflect the these development artifacts *design model* and *source code*. For this purpose, the DesignSpace offer an editor/visualization tool called the *WorkBench*, which engineers may use to access artifacts that otherwise might not be visible to them (e.g,. a software engineer likely does not use a CAD modeling tool and thus would not be able link code to artifacts from CAD drawings). The WorkBench provides tool-independent visualization and editing services to let engineers view and manipulate artifacts at will. This is not only useful for linking artifacts but also to provide domain-specific extensions that could not easily be modeled within an engineering tool. For example, engineers can use this editor to define equality links among the "arm length"-specific parts of artifacts in the three engineering tools discussed in Figure 1.

In this context, it is important to note that the DesignSpace does not "hard code" a specific metamodel/ontology. Rather, they may vary and metamodels/ontologies can be changed like any other artifacts. The artifacts in the DesignSpace are weakly typed to their metamodel artifacts to also support language/domain/metamodel evolution. Contrary to state-of-the-art, the DesignSpace stores both the artifacts and their metamodels. And an engineer may evolve these metamodels at will to support additional engineering knowledge such as links between UML and Java source code.

It is also important to note that the actual integration with the engineering tool through adapters is application specific and discussed elsewhere [3]. However, it should be mentioned that most engineering tools today allow live, programmatic access to its artifacts which we exploit to implement that tool-specific adapters. If an engineering tool does not support live synchronization of development artifact changes with the DesignSpace, the synchronization could also be implemented through periodic parsing of the files it stores. And it should be note that the level of abstraction and the granularity used for representing development artifacts in the DesignSpace can be chosen arbitrarily. For example, a Java class file can be represented in the DesignSpace at a course granularity by a single node, or at a fine granularity by mirroring the class' syntactic structure.

Figure 2 depicts the DesignSpace in principle. On the left, we see the engineers and their engineering tools which they use locally on their work stations. Communication with the DesignSpace is implemented in a REST-full enterprise architecture and we see that the DesignSpace's Data Service mirrors the development artifacts of the engineering tools. The arrow between the 'Requirements' and 'Code' artifacts represents a link that has been added by an

engineer using the Editor/Visualizer depicted in the bottom left. Note that arbitrary complex artifacts may be added in this manner.

4.2 Engineering Services

Figure 2 also depicts an overview of the engineering services provided by the DesignSpace. For example, recall the equality links among the "arm length"-specific parts of artifacts in the three engineering tools discussed in Figure 1. Such equality links can be defined by engineers using the WorkBench and such equality links can be automatically checked for correctness/consistency using the *Consistency Checker* service provided by the DesignSpace. The consistency checker reacts to any predefined constraints which could be a simple equality constraint or more complex well-formedness rule across multiple artifacts (we use the Model/Analyzer consistency checker here because of its fine-grained, incremental approach to changes and the corresponding scalability [4]). For example, a more complex constraint would be to keep track of the correct robot's weight for the control software since the weight is an aggregate of many other artifacts modeled in different tools (the weight is needed to correctly control the servo motor acceleration).

The *Subscription-Based Change Notifier* (depicted in Figure 2) informs engineers or tools about changes to artifacts they are interested in; e.g., the project leader may want to be informed if critical mechanical changes are made in a CAD drawing as they tend to affect many parts of a system. The *Trace-Based Change Notifier*, on the other hand, informs engineers about changes on artifacts that are linked via a trace link. A trace link is a special kind of link much like the equality link above. Engineer may use a trace to, say, keep track of where a requirement is implemented. If the requirement changes subsequently then all engineers are notified whose artifacts were traced to that requirement. Since the DesignSpace keeps track of who did what and how, it is straightforward to identify engineers who are affected by a change. Note that the DesignSpace, by default, does not automatically create links or traces but helps maintain and reason about them—capturing traces is part of the design process that is enabled by the DesignSpace. However, *Refactoring and Transformation* services may be used to generate traces or other artifacts automatically. The DesignSpace's virtual access to all engineering tools allows refactorings and transformations to span across tool boundaries. For example, consider that in Figure 2 we use the term "length" to denote the arm length of a robot. This term may be ambiguous and a global replacement of the term "length" with "arm length" would require an coordinated effort of multiple engineers using multiple tools. Instead, the refactoring service on the DesignSpace could do this more elegantly and the changes are then synchronized back to the engineering tools. Automatic, bi-directional synchronization is supported with most integrated engineering tools. However, we believe that engineers should be asked for permission prior to performing automatic synchronization of artifacts in tools.

Finally, the DesignSpace provides a *Syntax and Semantics Checker for Co-Evolution*. Co-evolution ensures the conformance of artifacts to their language (e.g., metamodel) after language evolution. Most engineering tools already enforce their own language and this service is thus only needed if the engineers wish to diverge from a tool's language or, more significantly, if they find it useful to define a domain-specific model (perhaps one that spans across artifacts from multiple tools). In case of the robot example, such a domain model exists, describing the main components of a robot (not depicted in the figure due to brevity). This service then ensures that changes of this model do not break the model's conformance with the desired language (e.g., well-formedness), thus avoiding errors being introduced. This conformance checker is also useful for additional knowledge added by engineers. For example, equality links are only then well-formed if they link two same-typed artifact properties. It would be invalid to, say, link a natural number with a Boolean.

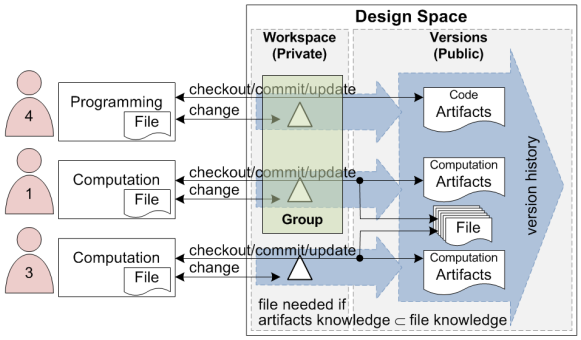

Figure 3: DesignSpace Collaboration Overview.

4.3 Collaboration Services

Collaboration is an integral part of the DesignSpace. In a cloud-based engineering environment, however, the preferred style of collaboration may vary significantly depending on project specifics such as project complexity, development process, team size, or personal preference—the preferred style may even change during a single project. For example, engineers may prefer to collaborate closely by following a blackboard-style approach in which all artifacts are shared and kept in a public, version controlled space that is accessible to all (with the proper access rights), similar to Google Docs. But engineers may also choose to not share certain artifacts (temporarily) which are then kept in private work spaces, similar to SVN where private changes are not publicly visible until they are committed. To support any kind of collaboration, in the DesignSpace there is thus one public work space and potentially many private work spaces. For any change that is about to be made, engineers can decide flexibly and on demand whether they prefer to work in the public or a private work space.

The DesignSpace approach supports versioning at the finest-most level of granularity: each node and each edge is versioned separately. This makes the version control mechanism of the DesignSpace quite simple and efficient. If, for example, the name of a method in a Java class is changed then committing the change leads to a new version of the class name in the DesignSpace. This fine-grained versioning is also resulted by the desire to support fine-grained change notifications and change propagation as discussed above.

As depicted in Figure 3, the DesignSpace does not only provide publicly visible *versions* but also private *work spaces* (versions not committed yet, representing an engineers ongoing work). The private work spaces reflect the current, ongoing look at what engineers are doing. There is exactly one work space for every engineer and the tool that is used. Reflecting both public and private knowledge, the DesignSpace can provide unified and integrated engineering services across all artifacts; e.g., to inform an engineer about inconsistencies the private changes would cause prior to committing them. This is particularly interesting for cross-artifact, and thus often cross-tool, inconsistencies which would not be detectable within one's own tool. Consider, for example, that the mechanical engineer makes changes which lead to a new arm length. In a SVN style approach, such a change would remain invisible

to the software engineer, even after the change was committed (i.e., because the software engineer would be unaware of this change until he/she chooses to update). This may not be useful. To avoid this, the DesignSpace can allow the consistency checker to work on the latest public knowledge, which means that the software engineer would be instantly notified as soon as the mechanical engineer commits the change. Indeed, the DesignSpace also allows for changes to become visible to others even before committing them into the public work space. This is realized via *work space groups* (arbitrary and even overlapping combinations of private work spaces) which represent a kind of joint, but still protected, work space in which a group of engineers can work together and see their work prior to it becoming public knowledge. If the mechanical engineer and software engineer where to form such a group, the software engineer would be notified of the arm length inconsistency at the moment the mechanical engineer makes the change, even if the change is not publicly visible. Different modes for such work space groups are conceivable, from sharing all the knowledge between private work spaces to more restricted forms, which will be part of our future work.

5. PROTOTYPE IMPLEMENTATION

To date, the core data and engineering services of the DesignSpace have been implemented.[1] These services include: i) data storage mechanisms that allows for cloud-based mirroring of arbitrary development artifacts, ii) versioning of artifacts (with conflict detection and support for various types of collaboration), iii) traceability, iv) consistency checking, v) subscription-based change notification for artifact changes, vi) trace-based change notification for artifact changes, and vii) an editor with basic visualization capabilities. Moreover, adapters and plug-ins are available for various engineering tools to synchronize their artifacts automatically with the DesignSpace. Currently, the following tools are already supported: ProEngineer, IBM Rational Software Architect, and Microsoft Excel.

6. APPLICATION AND VALIDATION

The DesignSpace is an ongoing project. However, it is already being applied within the software engineering domain and the mechatronics domain.

6.1 EPlan

For this application of the DesignSpace, the tool EPLAN Electric P8 for the development of of electrical models and the well-known Eclipse IDE for source code development were integrated with the DesignSpace though the means of tool adapters. Traceability between EPlan models and source code were established by engineers using the DesignSpace's own editor tool. Consistency between the electrical model and source code was checked based on a set of user-defined, domain-specific rules.

6.2 ACCM Robot Arm

The DesignSpace has also been used in the mechatronics domain as a platform for designing a robot arm. The project involved various kinds of artifacts. For example, mechanical calculations were provided in the form of multiple Excel spreadsheets. UML models of the robot arm were built with the IBM Rational Software Architect. Moreover, there were 3D models of the robot arm built with CAD tools as well as simulation models built with Matlab. All artifacts were represented in the DesignSpace and traceability between the artifacts was established. The DesignSpace's data services were used to check consistency among artifacts and to notify engineers about relevant artifact changes.

[1]Prototype available at `isse.jku.at/tools/dsspc/xadr.zip` (pw: dsisse).

6.3 ACCM Visualization Experiment

In the third major application, the DesignSpace was used as infrastructure in a controlled experiment with students. Different development artifacts (requirements, mechatronic design models) were stored in the DesignSpace and traces between those artifacts were established by domain experts. Students were then asked to perform defined refactorings, using different visualizations of the artifacts and the traceability (i.e., a matrix and a graph). To perform the refactoring, students used the DesignSpace's viewer and editor tool. Even though the focus of this experiment was to evaluate different visualization approaches, the DesignSpace provided essential functionality for linking distinct development artifacts and providing the data to be visualized.

6.4 Summary

The presented case studies demonstrate the general applicability and also the usability of the DesignSpace. Preliminary performance evaluations also indicate that it scales well with increasing numbers of mirrored artifacts. Incremental feedback about, e.g., version conflicts or change impact can be retrieved within milliseconds.

7. RELATED WORK

The DesignSpace provides a flexible engineering environment for efficient artifact and knowledge sharing. Indeed, there exist various tools and approaches that already provide parts of the DesignSpace's functionality such as artifact linking or versioning. For example, IBM RELM [5] allows arbitrary development artifacts to be stored, linked, and queried for reasoning. GME [6] supports the definition and subsequent use of languages and metamodels. Subversion [7] or Git [8] allow for different kinds of checkout-change-commit workflows with private working areas whereas Google Docs allows for blackboard-style, live collaboration. However, there is little compatibility between these tools and approaches—once a tool or approach has been chosen to be used in a project, the style of collaboration and the kinds of artifacts are determined. Later changes are costly and require much effort. The DesignSpace is a novel engineering environment that supports the combined set of workflows and kinds artifacts supported by the diverse existing tools and approaches—the actual workflow and artifacts to work with can be changed at all times without the need for changing employed technologies or tools.

Part of the vision of the DesignSpace is to allow the user to model an arbitrary number of domain models and models conforming to them, which in turn might also represent domain models (multi-level metamodeling). There exist various approaches that address this issue and furthermore also metamodel and model co-evolution—as models might no longer conform to metamodels if the metamodel changed – (e.g., [9, 10, 11]). Yet, all of them face some restrictions e.g., GME [11] uses a sophisticated, yet fixed, set of concepts to describe metamodels, which could lead to restrictions. Furthermore, constraints defined on metamodel elements must also co-evolve after metamodel changes (e.g., [12]). The DesignSpace keeps models and metamodels separate—even if the metamodel changes, the engineer is not required to update the model – indeed it may still be consistent with the older version of the metamodel.

Another approach to modeling are ontologies (e.g. RDF), which are used to represent domain knowledge in Semantic Web, knowl-

edge management systems or E-commerce. Those ontologies are typically developed and controlled in a distributed and collaborative fashion (e.g., OntoEdit [13]). Several works have been conducted on reasoning [14], versioning [15] or comparing versions [16] of ontologies. Nevertheless, a constructed ontology only provides syntactical consistency, its contents still can be semantically inconsistent. Evolution and traceability, even though there exist implementations providing both, are not innately supported. Therefore, the ontologies reach the same level of sophistication in (meta)modeling as the DesignSpace wants to support but lacks in other ways such as collaboration.

Neither of the previous mentioned work fully addresses the collaboration. Typically collaboration in software development—especially for source code—is often centered on sharing files which are organized by version-control systems like Subversion [7] or Git [8]. Versioning systems enable distributed collaboration on files, more fine grained control about what information to share with others is not possible. Likewise it is not (easily) possible to link artifacts from different files or provide the kinds of engineering services we suggested. Only file-based evolution and simple file-based traceability are supported. The DesignSpace contribution here is a finer grained handling of those files with the benefit that its services are now able to mirror those of engineering tools and beyond—enabling a cross-tool engineering infrastructure that is being used live.

A similar approach in tool integration is the Open Services for Lifecycle Collaboration (OSLC) [17]. The OSLC is a set of specifications to allow to integrate development lifecycle tools. Based on a core specification, domain specific specifications (e.g., configuration management, quality management, requirements management and architecture management) are built on top. The core specification only consists of standard rules for usage of HTTP and RDF and is not designated to be used on its own. OSLC integrates tools based on RDF and HTTP, allows tracing among RDF resources, modeling as so called resource shapes and not only data integration but also user interface integration of different tools. However, it does not provide versioning of artifacts, co-evolution of metamodels, the metamodeling activity itself or consistency in the face of artifact evolution.

The DesignSpace also relates to many other works on consistency checking, traceability, etc. However, since it does not further their respective states of the art (e.g., the DesignSpace consistency checker is a state-of-the-art checker), these are not discussed here. Also note that the DesignSpace does not automate the linking of artifacts (traces, equality links, etc.). Engineers are expected to define those manually using the infrastructure it provides. Existing state of the art suggests that there are ways of (partially) automating links (e.g., traceability in particular). This was not investigated thus far but is the spirit of the DesignSpace and could be integrated.

8. CONCLUSIONS AND FUTURE WORK

The DesignSpace's vision is to cover the various dimensions of engineering—*collaboration*, *(meta)modeling*, *consistency*, *traceability* and *evolution*—by integrating and publishing artifacts and providing additional services. The DesignSpace allows to verify the consistency of artifacts, trace artifacts to one another, transform artifacts, enable (meta)modeling in arbitrary levels of abstraction, and record the engineering history at the finest level of granularity. Future work will be to finish the implementation of our working prototype, substantiate collaboration forms realized through workspace groups, and derive further case studies to verify our approach.

Acknowledgments

The work was kindly supported by the Austrian Science Fund (FWF): P23115-N23, P25513-N15, and P25289-N15, and the Austrian Center of Competence in Mechatronics (ACCM): Strategic Research Grant C210101.

9. REFERENCES

[1] B. W. Boehm, "Software engineering economics," *IEEE Trans. Software Eng.*, vol. 10, no. 1, pp. 4–21, 1984.

[2] S. Group *et al.*, "Extreme chaos," *The Standish Group International Inc. http://www. standishgroup. com/chaos. html*, pp. 1–12, 2001.

[3] D. S. Wile, R. Balzer, N. M. Goldman, M. Tallis, A. Egyed, and T. Hollebeek, "Adapting cots products," in *ICSM*, 2010, pp. 1–9.

[4] A. Reder and A. Egyed, "Model/analyzer: a tool for detecting, visualizing and fixing design errors in UML," in *ASE*, 2010, pp. 347–348.

[5] IBM. Rational Engineering Lifecycle Manager. http://www-03.ibm.com/software/products/en/ratiengilifemana.

[6] Á. Lédeczi, A. Bakay, M. Maroti, P. Völgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai, "Composing domain-specific design environments," *IEEE Computer*, vol. 34, no. 11, pp. 44–51, 2001.

[7] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato, *Version control with subversion - next generation open source version control*. O'Reilly, 2004.

[8] J. Loeliger, *Version Control with Git - Powerful techniques for centralized and distributed project management*. O'Reilly, 2009.

[9] S. Fickas, M. Feather, and J. Kramer, "ICSE-97 Workshop on Living with Inconsistency, Boston, USA," in *ICSE*, 1997.

[10] L. M. Rose, D. S. Kolovos, R. F. Paige, and F. A. C. Polack, "Model Migration with Epsilon Flock," in *ICMT*, 2010, pp. 184–198.

[11] J. Davis, "GME: the generic modeling environment," in *OOPSLA Companion*, 2003, pp. 82–83.

[12] A. Demuth, R. E. Lopez-Herrejon, and A. Egyed, "Cross-layer modeler: a tool for flexible multilevel modeling with consistency checking," in *SIGSOFT FSE*, 2011, pp. 452–455.

[13] Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, and D. Wenke, "OntoEdit: Collaborative Ontology Development for the Semantic Web," in *International Semantic Web Conference*, 2002, pp. 221–235.

[14] X. Wang, D. Zhang, T. Gu, and H. K. Pung, "Ontology Based Context Modeling and Reasoning using OWL," in *PerCom Workshops*, 2004, pp. 18–22.

[15] D.-H. Im, S.-W. Lee, and H.-J. Kim, "A Version Management Framework for RDF Triple Stores," *International Journal of Software Engineering and Knowledge Engineering*, vol. 22, no. 1, pp. 85–106, 2012.

[16] N. F. Noy and M. A. Musen, "PROMPTDIFF: A Fixed-Point Algorithm for Comparing Ontology Versions," in *AAAI/IAAI*, 2002, pp. 744–750.

[17] *Open Services for Lifecycle Collaboration http://open-services.net/*, Std.