Engineering Web Community Information Systems via Near Real-Time Collaborative Modeling Support

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Petru Nicolaescu
M.Sc.

aus Târgovişte, Rumänien

Berichter: Univ.-Prof. Dr. rer. pol. Matthias Jarke
Priv.-Doz. Dr. rer. nat. Ralf Klamma
Univ.-Prof. Dr. S.G. Stephan Lukosch

Tag der mündlichen Prüfung: 25. Juni 2018

Abstract

Driven by the emergence of new standards, protocols and architectural patterns, Web information systems are gradually shifting towards social ecosystems, featuring near real-time communication – synchronous, always without human noticeable delay though less stringent than real-time requirements in embedded systems – and collaboration support with few restrictions. Spreading from personal to professional and organizational settings, engineering such powerful, collaborative applications is difficult to achieve as it requires considerable know-how and is associated with high costs. Especially within small, niche communities there is a high need for rapid Web information systems development. For example, professional online communities of practice do not possess the technical knowledge to implement the specialized tools they need or restructure systems, without developers.

Combining novel synchronous Web-based collaboration techniques with conceptual modeling and model-driven Web engineering will radically improve this situation. This mixture can increase collaboration and awareness between stakeholders, lower the technological entry barrier in system engineering, foster rapid prototyping and ensure standard compliance.

This dissertation proposes a new, cyclic, Model-Driven Web Engineering approach that uses shared editing and near real-time collaborative conceptual modeling in order to create Web-based, social and collaborative community information systems. SyncMeta, our collaborative conceptual modeling approach and framework supports view-based modeling in the Web browser in synchronous, distributed environments. In addition to domain-specific visual modeling languages, viewpoints can be collaboratively defined on the metamodeling layer and instantiated as views within a model editor instance. For supporting SyncMeta's lock-free, shared model editing in highly scalable team sizes, we have created YATA, a novel optimistic decentralized concurrency control approach realized as a Javascript Web library named Yjs. This solves occurring conflicts in groupware scenarios and can be used very flexible to enable near real-time collaboration in existing or newly engineered Web applications. Using SyncMeta and Yjs, we have instantiated the Community Application Editor (CAE), that is used to model and define Web widgets and microservices as software components that together compose community information systems. CAE makes it simpler – using the modeling abstraction – for both developers and community members to contribute to system design and development.

The research approach of this dissertation follows the design science methodology and has been evaluated using various testbeds involving professional communities from the Technology Enhanced Learning, Cultural Heritage and Construction domains, in several user studies. All in all, we showcase the efficiency of collaborative conceptual modeling in agile model-driven scenarios and thus contribute to the engineering of modern, social Web information systems. We offer support for creating collaborative applications tailored for professional, online communities of practice.

Kurzfassung

Angetrieben durch das Aufkommen neuer Standards, Protokolle und Architekturmuster, verlagern sich Web-Informationssysteme allmählich in Richtung sozialer Ökosysteme und bieten "beinahe Echtzeit (NRT)" Kommunikation – synchron, ohne eine für den Menschen wahrnehmbare Verzögerung, jedoch mit weniger strengen Anforderungen als bei der Echtzeit Anforderung eingebetteter Systeme - und Kollaborations Support mit kaum Einschränkungen. Gerade in kleinen Nischengemeinschaften besteht ein hoher Bedarf an schneller Web-Informationssystementwicklung. Zum Beispiel verfügen professionelle Online "Communities of Practice" nicht über das technische Wissen, um spezialisierte Tools zu implementieren oder Systeme ohne die Hilfe von Entwicklern anzupassen. Die Kombination neuartiger, synchroner und Web-basierter Kollaborationstechniken mit konzeptioneller Modellierung und modellgetriebenem Web-Engineering kann die Zusammenarbeit zwischen Stakeholdern verbessern und die technologische Eintrittsbarriere zur Systementwicklung verringern und schnelle Prototypenentwicklung fördern.

Diese Dissertation schlägt einen neuen, zyklischen und modellgetriebenen Web-Engineering Ansatz vor, der kollaboratives Editieren und NRT-kollaborative konzeptionelle Modellierung verwendet, um Web-basierte, soziale und kollaborative Informationssysteme zu erstellen. SyncMeta, unser kollaborativer Ansatz und Framework, unterstützt die modellbasierte Modellierung im Webbrowser in synchronen, verteilten Umgebungen. Zusätzlich zu domänenspezifischen visuellen Modellierungssprachen können "Viewpoints" gemeinsam auf der Metamodellierungsschicht definiert und als "Views" in einer Modelleditorinstanz instanziiert werden. Zur Unterstützung der gemeinsamen Modellbearbeitung von SyncMeta in hoch-skalierbaren Teamgrößen haben wir YATA entwickelt, einen neuartigen optimistischen Ansatz zur dezentralen Steuerung des gemeinsamen Zugriffs, der als JavaScript Webbibliothek namens Yjs realisiert wurde. Diese löst auftretende Konflikte in Groupware-Szenarien und ist flexibel einsetzbar, um eine NRT Zusammenarbeit in bestehenden oder neu entwickelten Web-Anwendungen zu ermöglichen. Mithilfe von SyncMeta und Yjs haben wir den "Commnunity Application Editor" (CAE) instanziiert, mit dem Web-Widgets und Microservices als Softwarekomponenten modelliert und definiert werden, die zusammen Community-Informationssysteme bilden. CAE vereinfacht es - durch Verwendung der Modellierung Abstraktion - sowohl für Entwickler als auch für Community-Mitglieder, zum Systemdesign und zur Systementwicklung beizutragen. Der Forschungsansatz dieser Dissertation folgt der Design-Science-Methodologie und wurde in verschiedenen Benutzerstudien mit verschiedenen Testeinrichtungen, an denen professionelle Communities aus den Bereichen Technologie-gestütztes Lernen, kulturelles Erbe und Bau beteiligt waren, evaluiert. Zusammenfassend demonstrieren wir die Effizienz kollaborativer, konzeptioneller Modellierung in agilen, modellgetriebenen Szenarien und tragen zur Entwicklung eines modernen und sozialen Web-Informationssystems bei. Wir bieten Unterstützung bei der Erstellung kollaborativer Systeme für professionelle Online-Communities of Practice.

Acknowledgment

Thinking about the past years, I realize how many persons helped, guided or inspired me and how much I owe them. First of all, I would like to express my sincere gratitude to Prof. Dr. Matthias Jarke for his guidance and advices and for giving me the opportunity to pursue the doctorate at the Chair of Information Systems and Databases at the RWTH Aachen University. It was a pleasure and honor to work under his supervision. His research experience and his leadership decisively helped me in finishing the dissertation. I am equally grateful to Dr. Ralf Klamma, for giving me the chance to be part of his team and for acting as "in loco parentis" during my studies at RWTH Aachen University. His frequent counsel and encouragements not only shaped my work but also had a big impact on my start and evolution as a researcher. I am thankful for having him as my supervisor and for the constant inspiration he has offered me. Under his protecting wing, our collective evolved into a true community. We tied strong friendships which will hopefully last a life time. I am also grateful for his trust, for the freedom I have received and for all opportunities in the international research context offered throughout the years. In addition, I thank Prof. Stephan Lukosch for accepting to be part of the dissertation committee and for the very interesting discussions we had in the rather short time we met.

My research and the current form of the dissertation was greatly and decisively impacted by the views and advices of Dr. Michael Derntl. It was a pleasure for me to know him and work together and I am grateful for all the refreshing discussions and ideas we have exchanged, and all the help he offered me. Moreover, it is an honor to have his friendship. His view on the world was always an example and very often enlightened my mood.

Special thanks, I owe to my Mathematics teachers, Mr. Aurel Dobrin and Ion Solomon, and my violin teacher Justin Gropescu. Their dedication and efforts initiated me into hard work. They also opened in me the desire to explore, seek performance and managed to introduce and guide me into the world of mathematics and arts, with a major impact on my life.

During the PhD years, I was fortunate to be part of a team that made the time spent working worthwhile. I would like to thank G. Toubekis, Dr. Y. Cao, Dr. A. Hannemann, Dr. Z. Petrushyna, Dr. D. Kovachev, Dr. M. Derntl, Dr. M. Kravcik, Dr. K. Rashed, Dr. M. Pham, I. Koren, Dr. M. Shahriari, P. de Lange, K. Neulinger, et al. We traveled together to wonderful places, established nice collaborations and friendships, learned from each other and finished successfully several big projects. I will never forget the inspiring discussions and the evenings spent at the chair together with Dejan and Pham, the meetings at Yiwei, nor the fun chats with Milos and Michael, the former's patience and calm and the latter's sense of humor. I am thankful for the time spent at the chair or during traveling together with István, Dejan, Milos, Yiwei, Dominik, Peter and Michael and all the experience gathered from the project work and working with trendy technologies in various contexts. I am also grateful for the big amount of support which I have received in the last two years of the dissertation. Special thanks go to my office colleagues, Anna and Peter for sharing wonderful memories together. In the last years, a strong friendship grew between Peter

and me, resulting also in joint ventures such as growing chillies, parties and occasionally practicing sports. He also offered valuable help during the last stages of the PhD, with proof reading, rehearsal listening and document printing. Finally, I am grateful to Daniele Gloeckner, Claudia Puhl and Gabriele Hoeppermanns for helping me with all workplace organizational activities and to Reinhard Linde and Tatiana Liberzon for their valuable help with administrative tasks and for always providing a listening ear and a shoulder to cry on.

My work was greatly improved through the proficient work of the students I have collaborated with and who I was very lucky to (co)advise. I am deeply grateful to Mario Rosenstengel, Kevin Jahns, Peter de Lange, Stephan Erdtmann, Volkan Günal, André Tebart, Aarij Siddiqui, Sadik and Bujar Bakiu and others for all their dedication, independent and hard work and wonderful results achieved together. I have learned much from each of them and I am proud to have had them in my team.

I am equally grateful to my friends, for their support and for making my life easier and happier. I would like to thank Dejan and Alexandru, my best men and Yiwei, for all the emotional and protective support offered, as well as being role models worth following. I would also like to thank Sten, Adeline, Nils and Simona, Gökhan, Zied, Marina, Victor, Marion, Nele and Ina, Svenja and Stefan, Cornelia, Karl and Ruth.

My personality, education and well being I deeply owe to my parents, Georgeta and Petre Nicolaescu, whom I would like to thank for their constant love, dedication and self-sacrifice. Through their efforts, I had the privilege to grow in a protected environment where they always provided everything I needed. They have been a source of inspiration, wisedom and motivation. Furthermore, part of what I am today is also due to my aunt and oncle, Dinu and Felicea Lucaci who supported me unconditionally. I am deeply grateful for hosting me during the entire Bachelor period in Bucharest and for offering me their love my entire existence. Equally, Ileana Hada, my mother-in-law, always considered me as her son and offered me continuous love, comfort and trust, for which I am deeply grateful. I also thank Ioan Dragomir, my father-in-law, for his support. Furthermore, I would like to thank my "sister" Irina Cristian, Iulian, Diana and Robert Cristian and our dear friends Mariana, Matei and Andi Stoica, to whom I am deeply indebted and who were near me since I was born and provided the needed mental support during happy or tougher moments. The same holds for our extended family members Angela and Vasile Cosac, Mariana Fratila, Ina Gherghel. I am very very lucky to have a wonderful, loving and caring family.

Finally, with all my heart, I wish to thank my dear wife and better half Ana-Maria-Cristina, for her love, dedication and encouragements. My best friend and always by my side, her counsel and care helped me through every situation and I would not have even started this dissertation if not for her support. I am deeply grateful and eager for the challenges awaiting ahead, on the path we are walking side by side for the upcoming decades. During the doctorate years, we have witnessed wonderful events together, such as travels, our wedding and the birth of our son, Erik George. Last but not least, Erik offered me an extra motivation and brought lots of joy into our lives during the past two years, for which I am incredibly proud and happy.

To my mother, whom I dearly miss, for shaping and inspiring the man I strive to be; to my father, for his wisdom, self-sacrifice and continuous efforts directed towards me; to my wife and soul mate for her pure love and unconditional support.

Contents

The journey of a thousand miles
begins with one step.

Laozi (604-531 BC)

Chapter 1

Introduction

Distinguished by collaboration and interactivity, Web 2.0 is being increasingly adopted into multiple domains, impacting personal and corporate spaces alike. Transforming from traditional hyperlink applications, coined at the beginning of 1960s by Ted Nelson and Doug Engelbart, many Web 2.0 information systems are seen nowadays as "social software". Supported by a plethora of emerging technologies and standards, this is designed for hosting, creating and consuming content [Tim05, Kla15]. The results include multiplatform responsive applications, social networking sites, content management systems (CMS), multimedia platforms and communication and collaboration tools such as chat applications, wikis, blogs and forums [KW07a] [KJ08].

Building on social Web features, platform companies such as Facebook, Google, Twitter, Wordpress, Wikipedia, LinkedIn became very popular and successful. Furthermore, relying on informal communication and the collaborative generation of content, they open a wide range of ideas and socially-engineered knowledge. Application programming interfaces (APIs), libraries and even programming languages evolved in order to support developers to build such online platforms and applications. In 2017, Gartner writes about the API economy as a top strategic technology trend, turning businesses and organizations into platform providers. These technologies also lower the entry barrier for Web users, allowing them to produce and communicate content. CMS systems make it easy to build Web pages even without technical knowledge. Such pages can be directly used to share and comment textual content, pictures, images and videos.

However, beyond such systems, sustained by infrastructure paradigms and architectures such as cloud computing and peer-to-peer (P2P), Web-based information systems that fulfil communication and collaboration requirements of individual users or user groups receive lately increased attention. Moreover, agile Web development methods and the new synchronous and asynchronous communication means lead as well to major changes in the practice of professional communities [Wen98] situated in the long tail [And08], in terms of adopted information systems and usage of new socio-technical means. Existing research shows that knowledge in Communities of Practice (CoP) [MF00] can be supported

by means of social software (such as discussion groups, chat rooms) and leverage the incentives of its members to a participatory behaviour. Communities from domains such as technology enhanced learning, cultural heritage, healthcare, architecture or construction started to use social software on the Web in order to share ideas, collaborate, improve workplace processes and – most important – evolve the shared knowledge of the community [KW07a]. Even though CMS-like solutions are very convenient for content production and the resulting artifacts are easy to customize in terms of appearance and features, supporting the heterogeneous needs of professional CoPs cannot be so easily achieved using only such means.

While trying out new promising tools, communities can seldom find one that fulfils all their requirements (usually only some requirements are fulfilled). Moreover, in case some tools are adopted by a community, the problem of tool adaptation arises, often due to the lack of interoperability. Therefore, CoPs need specialized services that can fulfil their needs [Wen10, EDZC09] and cope with challenges such as transcription, localization, addressing [JK02, KSCJ06] and continuous change in requirements due to technical, societal, political or regulatory pressures. In this context, there is a need of customization in terms of collaboration adoption and software development methodologies which can deal with the lack of simplicity and flexibility in the software landscape of communities and that can also enable – as developers often lack in such long tail communities – non-technical community users to be able to customize applications and services that can augment and serve the needs and the practice, beyond simple text and multimedia sharing solutions offered by traditional Web 2.0 solutions.

Together with a decrease in cost and the high availability of computing power and mature standards/protocols, the rise of the Web 2.0 shifted best practices during the last decade also in the world of software development. Driven by the high demand of developers and the large gap between existing ones and the huge Web user basis, agile methodologies where end users are actively and continuously involved in application design and development (such as end user development (EUD) [WJ04, MKB06, Pat13], participatory design [KB98]) gained momentum. This is of particular importance in Web information systems engineering field. Here, end users are the ones permanently requesting and contributing to changes and evolution of applications, driving frequent software updates, component refactoring and also innovation, especially in the long tail.

Together with suitable platforms, agile Web information systems development represents a suitable driver for rapid uptake of technological advances and the re-design of software landscapes into agile modern systems [RCMX06]. Especially for continuously evolving Web applications, contributions from a wide variety of geographically distributed stakeholders, their involvement in negotiation and impact analysis from different perspectives and the rapid prototype generation from specifications are of great importance. From a social perspective, requirements elicitation and agile development processes are often the result of collaboration between team members, which may not be geographically co-located, yet are accustomed from their usual Web tools that everything has to happen in near real-time

(NRT), "almost always without human noticeable delay though less stringent than real-time requirements in embedded systems".

While literature on groupware (CSCW) regards such synchronous collaboration as real-time – in contrast to asynchronous systems such as email –, the embedded systems literature measures real-time responses in orders ranging from micro to milliseconds and require a response guarantee within such time frames. As on the Web response times are protocol-dependent and usually cannot be guaranteed due to geographical or latency boundaries and ranges can be expressed in up to hundreds of milliseconds, we consider such synchronous communication, where response times are below the user perceivable delay threshold, to be in NRT.

This dissertation investigates synchronous, NRT collaboration in the design and development of social Web applications for professional long tail CoPs. The focus is kept on development methodologies, practices and synchronous collaboration methods that can ease the collaborative creation of Web applications for both developers and non-technical community members. A view-based synchronous metamodeling approach is presented, which applies NRT collaborative Web-based conceptual modeling to a Model-Driven Web Engineering (MDWE) technique. The artifact realization is based on a novel algorithm and its implementation for shared editing on the Web, which targets to ease NRT collaboration adoption in Web applications. Our model-based, NRT collaborative Web engineering approach has been realized and validated through several frameworks and prototypes. Moreover, it has been evaluated in several domains such as Technology Enhanced Learning, Cultural Heritage, Healthcare and Construction, as part of various research and development European projects.

1.1 Example Scenarios

To reflect the current synchronous collaboration challenges in long tail CoPs, we shortly present here an overview of the application domain. Next, we detail two scenarios that reflect the need for reenginering and NRT collaboration adoption in professional CoPs and the opportunities opened by NRT view-based conceptual modeling for agile MDWE. Communication and collaboration established themselves as ubiquitous means used by CoPs for knowledge negotiation, strengthening of shared interests and progress towards shared goals. A prominent problem with collaboration on digital products —for instance in science, design or production— is that it is still being practised using cumbersome means like sending document drafts back and forth among collaborators. However, the collaboration needs often go beyond such traditional solutions (i.e., email, text editing or file sharing) and use the full flexibility and effort reduction leveraged by modern Web collaboration technology. There is often the case that workflows and practices have to be supported using integrated information systems, where collaboration is inherently enabled. Such systems have to be developed rapidly and also be maintainable, as the lack in developers

leads to a lack of technical knowledge inside the communities. Moreover, even though technology and methods for system creation already exist, current solutions often lack applicability inside professional CoPs. This can be solved by methodologies and support for developers and non-technical community members alike that simplify the development of collaborative information systems, such that technical solutions to evolving problems and needs can be rapidly developed and implemented, in an agile setting. Gathering several people distributed among various geographical locations in a single, shared Web "space" for NRT collaboration is the better option: collaborators do not need to install any software (i.e., most devices have Web browser capability), all participants can get immediate feedback on self and other's actions, communication can be done synchronously, etc. The need of NRT collaboration adoption, rapid prototyping and application reengineering can be exemplified in the technical support existing and needed for the co-creation of formal models of a learning design process.

Professional communities of learning designers did not benefit from modern Web technologies in their practice. This has happened despite the fact that the authoring of learning design models is typically a collaborative process involving various stakeholders such as teachers, students, instructional designers, education managers and others. For these communities, IMS Learning Design (IMS LD) is the available formalization for the specification of learning design models. In their state of the practice, learning designers use browser-based authoring tools for IMS LD (for example Web-Collage [HLVA+06]) and desktop applications that allow import and export of models (for example Reload [Reu04]). Each designer performs the collaborative authoring separately, using import and export functionality. In order to collaborate with other designers, sharing of the models is often executed via email exchange. Even though this communication means leads to delays and is prone to misunderstandings, the community members do not possess the resources to create a modern collaborative environment that can solve their problem. Moreover, due to the specificity of the IMS LD models, they cannot use existing third party Web applications for this purpose. While similar problems have been already surpassed (in collaborative text editing applications), transforming domain-specific modeling tools into Web-based, collaborative ones for multiple users cannot be achieved by learning designers without developer support. This transformation also requires considerate implementation efforts. Furthermore, both proficient developers and domain knowledge are required to achieve such goals.

Fig. 1.1 showcases such a domain-specific scenario. By abstracting it, we specify the need for agile and collaborative development practices of information systems for such CoPs. Given a set of elicited requirements in a joint project (particularly, the creation of a run-time simulation environment for IMS LD models), multiple stakeholders (for example end users, software engineers, architects and developers) want to use agile methods to design and develop a Web information system. A first prototype should be released as soon as possible. The various stakeholders establish a remote meeting session, where they can agree on a specification. During this session, they use a collaborative modeling platform

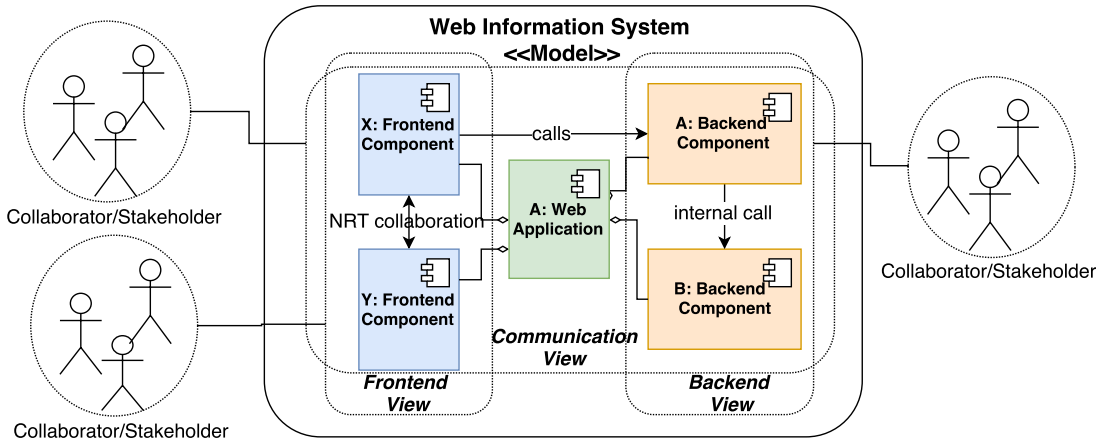


Figure 1.1: Scenario overview in a Model-Driven Web Engineering setting

to design the frontend and discuss and implement the needed functionality. Using this platform, models can be authored collaboratively in NRT using dedicated views between multiple users. As such, end users, designers and developers can co-design the frontend of the application using the frontend view. All modelers are aware of the remote actions of their collaborators and are connected using chat, video and voice communication channels. Assisted by developers, end users and designers add on the modeling canvas the frontend components which they envision in the final prototype. Developers and other stakeholders with technical knowledge model in NRT the backend of the information system (services, database, service interface, etc.) in a separate view on the model offered by the editor. Finally, to compose and release the final application, software architects are interested in a view which can depict the software components and the links between them. The Web information system can be constructed in this mashup view, where stakeholders can model the communication between components. The system is generated and deployed using the resulting model containing all elements authored in the three views.

1.2 Problem Description

Given the setting previously presented and the need to support non-technical members and developers in professional communities in the development of Web-based, collaborative applications, we investigate the methodologies, methods and the technical support which address this need. In a study performed in 2014 by the International Data Corporation (IDC), the total number of software developers in the world was estimated at about 18,5 millions, from which more than 7,5 million were considered to be hobbyists. A first

problem is therefore that even though an increasing number is targeting the mobile and Web application development, the gap between non-technical users and developers cannot be closed rapidly. The involvement of non-technical community members by means of active participation into application design and development (i.e. end user development EUD [WJ04], participatory design [KB98]) represent suitable drivers for technology uptake and the redesign of community software landscapes into Web-based, modern information systems. However, the right support for this transformation needs to be investigated.

Classical software development methodologies include end users and requirements elicitation phases in early phases or after specified iterations. In contrast, agile methodologies show that involving stakeholders – including non-technical users – more into design and development decisions can increase productivity and finally lead to a better software. A further problem here is that, in the scope of professional communities, it is not yet fully determined which methodologies and methods can be employed to increase agility, foster rapid prototyping and enable tailored development [SKW97] of community information systems. Generally speaking, there is a lack in methodological support for designing and developing Web application in a collaborative, synchronous manner.

Traditional methods and practices such as conceptual modeling remained far beyond the state-of-the-art collaborative techniques and Web protocols. Prilla et al. [PNH+13] note that even though collaborative modeling is very useful to support sense making, create a shared understanding and drive design changes, in practice, collaborative modeling is not used by or is not available for non-experts in organizations or other groups. The authors identify the lack of research on the interaction of various stakeholders during collaborative modeling as a main reason, together with missing insights of how such a Groupware activity can be used in practice together with non-experts. Web development methods which make use of model-driven approaches (WebML [BF14], UWE [KKZB08], OOHDM [SdM99]) or others (for example WebComposition [GWG97]) mostly consider data-driven Web applications or Web pages to design and generate conventional server-side solutions. The client-side interactivity which is very often present in synchronous, real-time Web applications is being omitted.

Development processes supporting synchronous collaborative work on the Web are still inefficient [HGSG12, Hei14]. Moreover, Web-based NRT shared editing solutions beyond text editing systems, such as Google Docs, are missing. It is very complex to design synchronous collaborative systems, or include NRT collaboration features into existing ones. A raising problem is how to reengineer many of the traditional Web applications (such as single page, single user Web applications using an n-tier architectural style with predefined page navigation and fixed user interfaces) for collaboration. As most of these Web applications show the lack of adoption for modern and lightweight collaborative environments, widget-based Web applications have become increasingly popular in the last decade for various purposes such as usability, spacing, capability of being embedded, personal computing environments, dashboards and distribution across devices. Finally, technical and algorithmic solutions do not consider P2P settings, increasingly available with

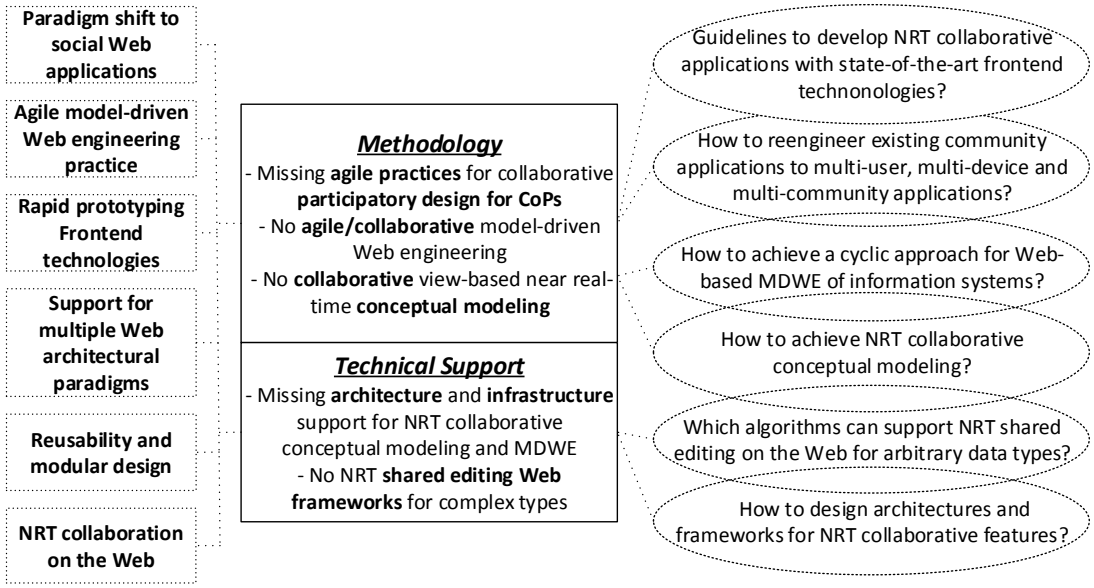


Figure 1.2: Hierarchy of main challenges, problems and subproblems tackled by this research

the emergence of standards such as WebRTC [BBJN13].

The above challenges are subsequently categorized therefore in two main areas (i.e. methodological and technical), as depicted in Fig. 1.2. The figure illustrates the key challenges covered in this work (cf. right-hand side), keeping into account the following detailed aspects (cf. left-hand side):

Web systems reengineering The transition and reengineering of desktop or single user Web applications to Web-based and mobile collaborative systems is in an early stage. While professional communities of practice can greatly benefit from NRT collaboration, many struggle with the adoption of collaboration technology [SVP05, BDV10, DKK+13, KNK14]. Beyond shared text editing solutions such as Google Docs or Etherpad, integrating NRT collaboration features into existing information systems or developing new ones currently requires a deep know-how. The Hypertext Markup Language 5 (HTML5) standardization has encouraged more interoperable implementations and allowed the rapid development of complex Web applications via the usage of APIs. This also turned HTML5 into a candidate for cross-platform mobile applications. Together with other Web standards and protocols, a new era of multi-user, multi-device and cross-platform Web applications and systems has emerged.

Agile model-driven Web engineering practices Although modeling is seen as a knowledge representation means in enterprise and agile environments [Kar15], the role of

traditional approaches such as conceptual modeling and model-driven approaches in developing Web 2.0 information systems for professional CoPs is still not clear. Here, there is a gap in investigating the impact which NRT collaboration has on conceptual modeling and how this can be realized both conceptually and technical. Moreover, given such collaboration on model elements as first citizen artifacts is possible, there is an obvious methodological gap in applying agile methods (requirements elicitation, rapid prototyping, unifying development and operations, continuous integration, etc.) for design and development of community information systems. In this thesis, we identify MDWE agile approaches based on collaborative modeling as the driving force for achieving a community information systems methodology for Web application development and for coping with the redesign of desktop-based, single user applications into collaborative, Web-based applications.

Rapid prototyping for frontend technologies Given the NRT requirements, in the information era developers need to be supported to build information systems faster, with a tighter integration of end users, make sure that their results are flexible in terms of scalability with the number of users, capable to run on multiple devices in the Web browser, fulfil the collaboration and communication needs and finally, cope with the context given by usage, devices and their capabilities. Considering the REST design principles as being an integrating part of the Web itself, research showed that these types of services perform better in different environments [RR07]. The need of proper user interfaces for such services having the capabilities enumerated above lead to the need of designing reusable frontends for rich user experience. Among the core features such as communication via service interfaces, frontends need to be able to also cope with P2P architectural paradigms, which are possible by protocols such as WebRTC and allow for direct NRT video, voice and data communication between browsers. Such protocols need to be exploited for realizing collaboration for community members and enriching the frontend's interactivity in multi-device, multi-user settings. Finally, it is not yet clear how existing community information systems artifacts can be reused in other communities, how such components can evolve without major impact on costs and workforce requirements. This is why reusability and modular design are important non-functional requirements that need to be considered by next-generation Web information systems.

Near real-time collaboration on the Web NRT shared editing solutions enable multiple users to collaborate on shared data [Gru94]. However, existing literature [WUM09] and technical realizations show that current shared editing approaches do not scale well with the number of users in pure P2P settings on the Web. Long-researched solutions in the CSCW literature mostly enable collaboration on linear data structures but are not designed for non-linear ones (such as graphs, custom abstract data types) or are designed for distributed server environments and the existing technical solutions do not target Web-based environments and development.

1.3 Research Questions

Considering the mentioned collaboration and technology adoption challenges identified in the community practice and the various interdisciplinary dimensions needing to be addressed in this dissertation, we formulate a set of core questions and subsequent, secondary points, contained in this research. Summarizing agile MDWE, NRT collaborative modeling and the technical aspects for shared editing on complex data structures, these are the following:

Research Question 1 (RQ1) - How to support CoP members in designing and building Web-based NRT applications?

- What methodologies or methods can be used for agile development and for transforming single-user applications into collaborative, NRT ones?
- Can Model-Driven Web Engineering enable rapid prototyping using social Web 2.0 technologies?
- How to design a development methodology and agile MDWE approach for enabling collaborative, modular, reusable, multi-purpose Web community information systems?
- How to decrease the entry barrier for building community tools?

Research Question 2 (RQ2) - Which technical and algorithm support is needed for enabling NRT collaborative editing features in Web applications?

- How to enable NRT shared editing beyond text and make it very easy to add NRT collaborative features to single-user Web applications?
- How can an overall architecture to support P2P and client-server NRT collaboration on the frontend be achieved?

Research Question 3 (RQ3) Can NRT collaborative conceptual modeling leverage CIS engineering processes?

- How to abstract the CIS design and make it customizable for heterogeneous communities?
- How to bridge the knowledge gap and offer a relevant abstraction for various CoP stakeholders? Can collaborative conceptual modeling enable agile participatory practices?
- How can traditional modeling challenges such as domain independent conceptual modeling, metamodeling, metamodel to model generation and view-based conceptual modeling be achieved from a NRT perspective?
- How to increase awareness and nudge modelers during NRT collaborative modeling?

1.4 Thesis Contributions and Research Context

This research investigates development methodologies, conceptual approaches, architectures and system perspectives with focus on frontend technologies, in the context of NRT collaboration for engineering Web information systems. We use the design science methodology [HMPR04], by seeking new and innovative digital artifacts and methods to expand the boundaries of member and community capabilities. Fig. 1.3 presents the main contributions and resulting artifacts, using the design science research methodology process model [HMPR04]. The targeted systems make use of powerful frontends, with enabled NRT collaboration and communication features. With know-how already existing on how to build powerful (cloud-enabled) server-side systems featuring the above characteristics (i.e. server-side paradigms such as cloud computing or P2P) and how to design services and APIs, we describe our technical contributions on architectures, systems and development models for building collaborative applications with an orientation towards frontends. We study the role that NRT collaboration on the Web has on conceptual (meta)modeling and model-driven approaches, applied in development lifecycles of information systems for professional communities of practice. By studies performed in cultural heritage, healthcare and learning professional communities, we propose a model-driven approach for designing community Web applications, which consist of componentized frontends and backends. We have revisited the conceptual modeling literature with the focus on Web-based synchronous collaboration and investigated NRT approaches using a state-of-the-art framework, called SyncMeta the role which NRT collaborative modeling has on such communities. The resulting architecture of our approach is based on P2P and client-server technologies and proposes a novel algorithm for consistency control in shared editing environments, which considers the heterogeneous needs of professional communities and Web developers.

The research was conducted in the context of several EU and RWTH Aachen University funded projects. The core parts of this dissertation, i.e. the methodologies, conceptual modeling and the NRT collaboration and shared editing support were developed within the EU-funded Learning Layers - Scaling up Technologies for Informal Learning in SME Clusters project (FP7-318209), partially using outcomes of the Responsive Open Learning Environments (ROLE)(FP7-231396) project. Additionally, our domain-independent metamodeling approach was supported by the EU LLP KA3 multilateral project METIS: Meeting Teachers Co-Design Needs by Means of Integrated Learning Environments. Among the project outcomes were SyncMeta, the NRT collaborative modeling and metamodeling framework on the Web and SyncLD, the first tool available on the market to support synchronous collaboration on IMS Learning Design units of learning. Further projects supporting the NRT collaboration, modeling and multimedia annotation use-cases presented in this dissertation were the Tempus SAGE and Erasmus+ Virtus projects and the RWTH-funded Exploratory Teaching Spaces Anatomy 2.0 project and the Hans Hermann Voss-Stiftung funded Digital ConProMa project.

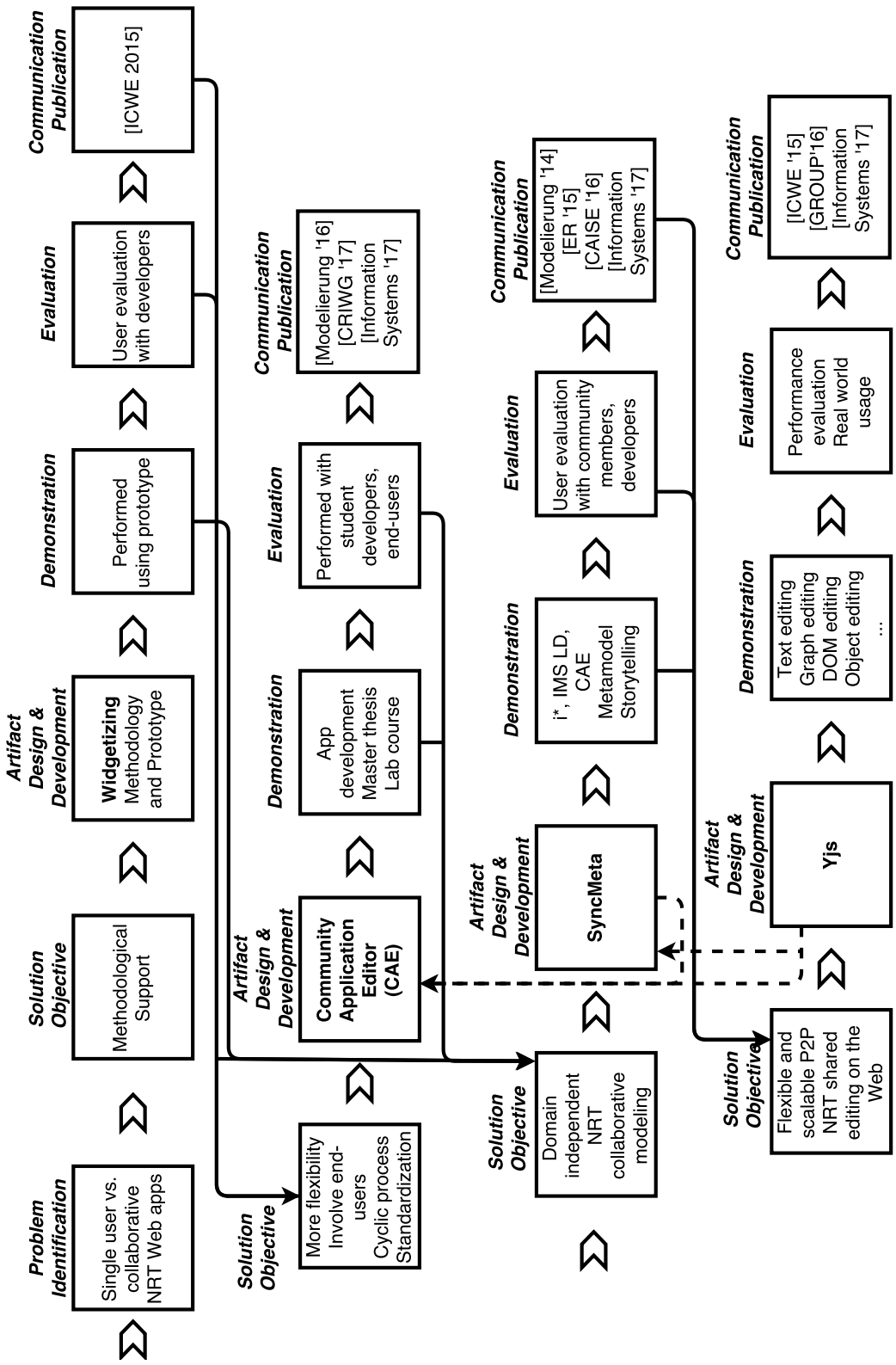


Figure 1.3: Research contributions and methodology according to design science process model

1.5 Thesis Outline

The rest of the dissertation is organized as follows:

Chapter 2 introduces the main concepts used in this dissertation, defines the used terminology and offers background information on agile model-driven approaches, conceptual modeling, NRT collaboration and community information systems. It also presents an overview on the state-of-the-art systems and approaches for each of the main contributions of this dissertation: MDWE, NRT shared editing on the Web and conceptual modeling. Finally, we detail the artifacts used and created in this work and introduce the test beds on which the work builds upon.

Chapter 3 describes the methodology proposed for developing agile model-driven Web community information systems. We start by introducing the requirements and challenges for achieving this and present the two main iterations which conducted to the current result. First, we present a study performed using widget developers from professional learning communities. Here, we also offer details on the evaluation and empirical studies performed based on our methodology. Based on the findings and requirements derived from this study, we then present the resulting general development cycle and the metamodel of community Web applications which is rooted into CoPs socio-technical practices.

Chapter 4 describes our contribution to the Web-based NRT collaborative editing using arbitrary data types. We present YATA, a novel algorithm from the conflict-free replicated data types family. Its implementation stands as technical support for our infrastructure and the prototypes built to showcase our conceptual findings.

Chapter 5 presents our findings on Web-based NRT collaborative conceptual modeling. We consider a domain-independent, non-locking collaborative generic visual modeling approach, based on metamodel to model generation. The collaboration experience during modeling is improved using awareness and guiding features. After presenting the main contributions in this domain, we offer a discussion regarding the modeling techniques and the shared editing algorithms which can be used to achieve such an approach technically.

Chapter 6 presents the technical prototype and the software architecture proposed in this work. We therefore describe the realization of the Community Application Editor, built on top of our conceptual modeling framework, using the NRT shared editing infrastructure. We also present the interplay and interactions which facilitate CoPs to build next-generation collaborative Web-based information systems. Focusing on user studies performed using the Community Application Editor, we showcase the feasibility of our approach and highlight the benefits of our research contributions in synchronous editing and conceptual modeling.

Chapter 7 presents several case studies which were used in developing community Web applications in professional communities from Technology Enhanced Learning and Cultural Heritage domains.

Chapter 8 discusses the results of this dissertation and offers an overview of open points and future research directions of our approach.

Great discoveries and
improvements invariably involve
the cooperation of many minds!

Alexander Graham Bell
(1847-1922)

Chapter 2

Research Context and Related Approaches

In this chapter, the technological and conceptual underpinning of NRT community information systems development, agile Web methodologies are presented, from the point of view of core existing research used in the dissertation for MDWE, conceptual modeling and NRT shared editing. We start by presenting an overall picture of the domains of interest and testbeds used in this work.

Fig. 2.1 summarizes the domains which are considered in this section, ordered chromatic by their corresponding domain/research field ("Information Systems", "Computer Supported Collaborative Work", "Web Engineering", "Technology Enhanced Learning"). We start by introducing CoPs and community information systems (CIS) and the theoretical underpinning used in this work, the ATLAS methodology [Kla10b]. The goals are to understand the formation, learning and knowledge building processes and the lifecycle of professional, on-site or online CoPs, that use technology to improve their practice. This is used as a basis for our CIS design and development approach. Further, agile methodologies for Web-based information systems engineering are introduced, with a focus on existing methods, systems and architectures. We also offer a technical overview of related work, focusing on the three main areas our contributions relate to, as mentioned above. We start by presenting the main MDWE methodologies and model-based platforms and systems already known in the Web Engineering literature. As collaboration is an important facet of this dissertation, we present the research landscape for NRT collaboration from the Computer Supported Collaborative Work (CSCW) domain [Gru94] by focusing on algorithms and systems employed for shared editing on digital artifacts, on the Web. We then present an overview of the conceptual-, meta- and view-based modeling concepts and the technical related work, which in the past decades only concentrated on single-user usage. Finally, we present the theoretical foundations on which our CoP-centric approach is based and the development testbeds used for technical/empirical studies and evaluation purpose.

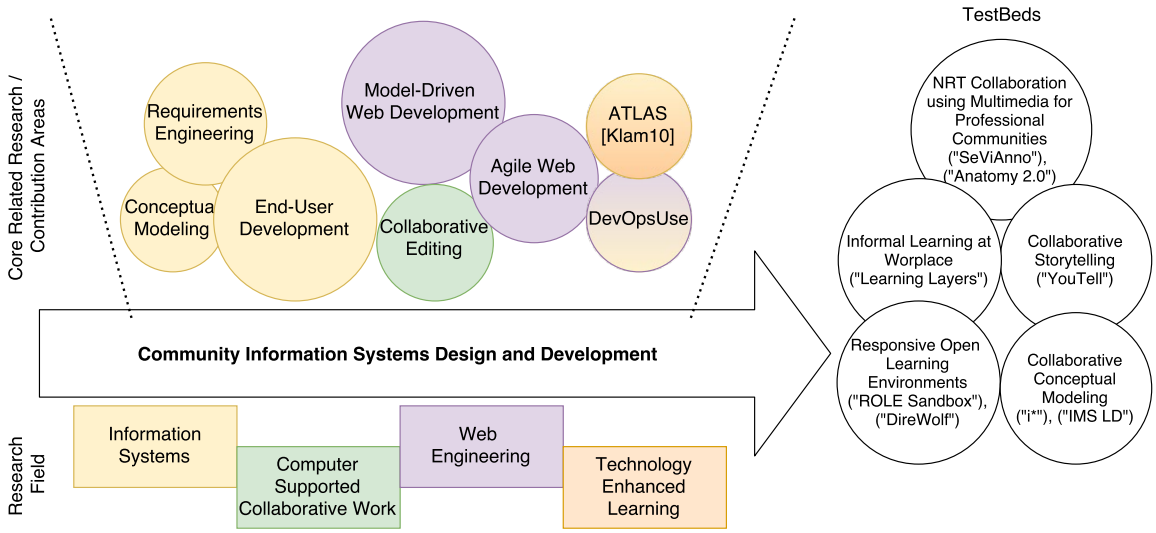


Figure 2.1: Lines of related research and testbeds

2.1 Information Systems for Long Tail Professional Communities of Practice

The term "long tail" is defined at the intersection of the CoPs theory [Wen98, Wen00] – which are active on the World Wide Web [And08] – with technology [Art11]. In order to highlight the importance of communities and digital artifacts as their support tools, we introduce the definition of a CoP and its main characteristics identified by various researchers in multiple social and ethnographic studies. We argument the involvement of community members in software development and the importance of (NRT) collaboration in such environments. For this, we relate to concepts and models used within our research group to support professional CoPs and build supporting information systems.

2.1.1 Community of Practice

The notion of CoP is part of the social theory of learning [SG89, BD91, Wen98, Wen00, Eck06]. Wenger defines a CoP as "groups of people who share a concern or a passion for something they do and learn how to do it better as they interact regularly" [Wen98]. This involves an evolving process of a group as a whole, via collaborative work and knowledge information sharing between community members. CoPs are a self-forming, voluntary construct and the result of work interactions between group members. Usually during their formation, they do not resemble organizational structures. CoPs are tightly bound to the notions of cooperation, collaboration, knowledge management and learning [BD00, BH02]. According to Wenger, the main characteristics of CoPs are the *domain* – a shared interest as identity providing factor –, the *mutual engagement* of members in activities or discus-

sions and *the practice* – members are also practitioners who develop a shared repertoire of resources such as experiences, stories, tools, etc. Therefore, social interactions in CoPs act as a medium for learning and build a sense of belonging and mutual commitment. Penelope Eckert [Eck06] identifies two key conditions of a CoP for meaning making: a shared experience over time and commitment to a shared understanding. This sense-making process is leveraged by collaboration between group members, which need to achieve a uniform understanding about the world around them and which involves relationships to other communities as well as an ongoing evolution of the shared practice. Later research [WWS09] shows that Web 2.0 social media technologies presence in communities fosters the formation, communication and collaboration.

Longtail CoPs. Web Science considers the Web to be a scale-free, power-law distributed network [Bar01]. On the Web, a vast majority of users are distributed in heterogeneous communities, forming a long tail distribution [Ren16, Kla10b], following Anderson's long tail economic theory [And08]. This predicted the shift from mainstream products and centralized markets into a big number of specialized, small markets, existing in the long tail of a power low distribution of demand. A representative example is Amazon's niche book stores, that created a consistent grow in gain during the 2000-2008 period [And08]. Same trends were visible for music on iTunes or movies on Netflix (95% of the entire movie collection was rented regularly by the platform's users). Furthermore, the intrinsic power of communities and the relations between global and localized system support were also recently commented by Facebook's creator, Marc Zuckerberg who in his "Building Global Community" post, talks about building and helping communities "from local to global levels, spanning cultures, nations and regions". This entails developing a social community-based infrastructure and in some sense signals the shift from mass to niche and from monoliths to customized and contextualized services.

Long tail CoPs are very dynamic with respect to their formation, adopted processes and lifecycle [LW91, WMS02]. CoPs involve members with various experience levels. They are subject to a continuous flux of idea exchanges and change in membership and member status, as new peripheral members join and existing ones leave the community. There are five steps identified in the literature characterizing CoPs development: potential, coalescing, maturing, stewardship, and transformation [WMS02, PL14].

In situated learning settings, sense-making and membership follows peripheral participation [LW91]. Through engagement in low-risk but important activities, novices, which are legitimate members part of the community periphery, become acquainted with the tasks, vocabulary and organizing principles of the CoP. They progress under supervision, slowly building their competence repertoire [Fox00]. Their learning is considered to take place in real-time. However, in [Wen98], Wenger introduces the notion of duality as the driving force for change and creativity. Meaning is considered to emerge by participation and involvement in a practice (participation - reification duality) and this itself, i.e., common activities, can be planned by certain designers or emerge through unplanned interaction between members (designed - emergent duality). In our work, we build on both concepts,

assuming that software artifact development is a practice that requires both collaborative learning and a strong involvement from the involved members.

Professional CoPs are communities active in certain domains, such as healthcare, cultural heritage, education, construction, etc. Studies deal with formation of CoPs in organizations, increasing the organizational performance [BD91, Wen98, LS01] , and formation of virtual CoPs within areas of interest such as the ones exposed above [APW03, Por04, DBJ05, HA13]. Developing professional learning CoPs is beneficial for building capacity and sustainable improvement [SL07]. The goals of professional communities are to augment the effectiveness of members as professionals and create value through an ongoing, reflective, collaborative, inclusive, learning-oriented, growth-promoting way [TL02, SL07]. A CoP can be regarded as a social learning system [Wen10], which exhibits characteristics of systems: emergent structure, complex relationships, self-organization, ongoing negotiation of identity and cultural meaning. In the same work, Wenger considers that "meaningful learning in social contexts requires both participation and reification to be in interplay. Artifacts without participation do not carry their own meaning; and participation without artifacts is fleeting, unanchored, and uncoordinated". In Fig. 2.2, Wenger summarizes a set of Web tools used and adopted by communities in their practice. We can observe the intrusion of

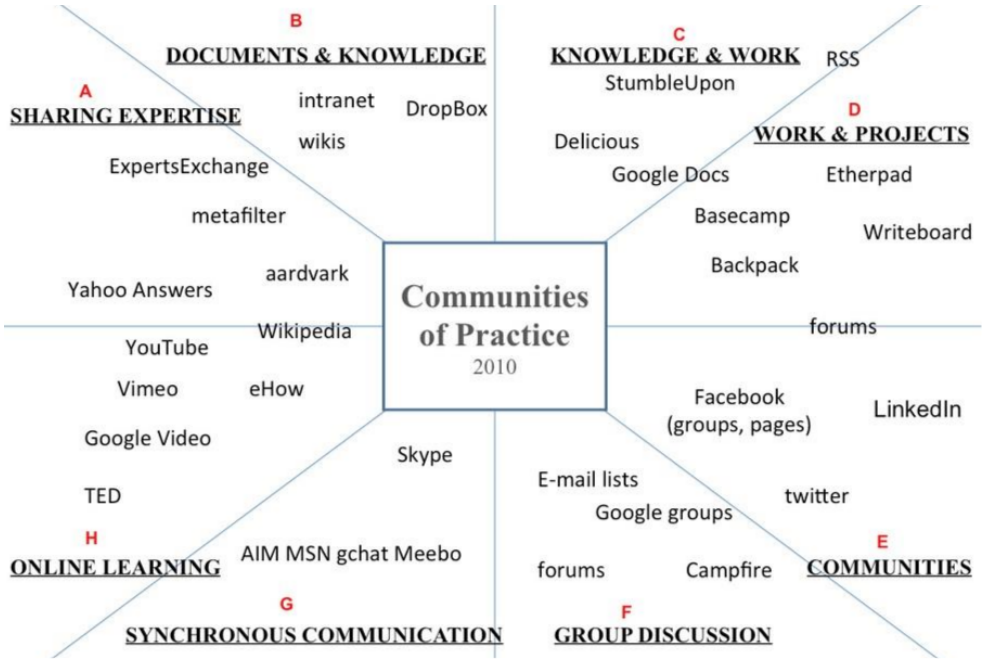


Figure 2.2: Tools that support CoPs [Wen10]

Web technology in many community activities. The Web shapes the communication, the knowledge sharing and creation, between community members. It extends the formation, membership and outreach of traditional CoPs, going beyond geographical borders and nations. Moreover, community dynamics, formation and life cycle are also influenced. It is

easy for new members to join a professional, online community. Having direct access to shared community knowledge, it is easy to directly search for existing information. Also, members can explore and analyse the structure of formed communities, make sure that the shared practice does not deviate from the main community interests, follow the other members, etc.

Thus, CoPs must benefit from the increased flow of information [WTWT15] and interconnectedness leveraged by technology and communication protocols. In 2015, in the same work [WTWT15], Wenger states : "The concept of community of practice is influencing theory and practice in many domains. From humble beginnings in apprenticeship studies, the concept was grabbed by businesses interested in knowledge management and has progressively found its way into other sectors. It has now become the foundation of a perspective on knowing and learning that informs efforts to create learning systems in various sectors and at various levels of scale, from local communities, to single organizations, partnerships, cities, regions, and the entire world." In this regard, we want to leverage the support for collaboration and software adoption beyond the scope of a single community. In order to achieve this, we study the structure and creation of CIS.

2.1.2 Community Information Systems

As explained above, the link between the use of technology and CoPs is very strong and it is clear that software artifacts have a great impact on the practice and innovation [Art11, LCK+16]. The opposite also holds, communities shaping the artifacts chosen for their practice. In terms of innovation, [Art11] distinguishes between four mechanisms: "1) novel solutions develop out of small standard engineering advances; 2) radical novel technologies develop through the process of innovation; 3) novel technologies develop by change in their internal parts or by addition to those parts through structural deepening; and 4) the wholesale emergence of new technologies that build and creatively transform industries over time (pp. 163-164)".

All software artifacts used by a (professional) CoP in their practice, for collaboration, knowledge management, coordination, etc. form a CIS. Their purpose is to improve effectiveness and efficiency within the community [HMPR04, HC10]. Information systems are defined in the literature as the bridge between humans and machines [IL84], that includes software, hardware, data, people, processes, interactions, boundaries, etc. [SMB95]. The CIS may evolve indefinitely as long as the community is active, as the composing systems are being updated, changed or terminated. Because of the direct connection between the software artifacts and the community practice, these are usually changing from inception to old age, depending on the needs they fulfil. The CoP principles explained before such as duality, legitimate peripheral participation, learning, collaboration, etc. also impact the design and usage of CISs. Examples include non-functional requirements, such as scalability, as the communities are highly dynamic and the number of members usually greatly varies between different, heterogeneous communities. Also, developers or members

with technical knowledge are not always part of a community, or the system development needs to be well coordinated in order to function properly. Community members may act as various stakeholders, from developer, designer to end user and therefore the need for collaboration and coordination is even greater than in usual software development processes (for example in organizations). The various roles are analysed from the perspective of supporting research in CoPs in [Kla10a]. In contrast, we focus here on the support of the collaboration between community members in designing and realizing their own Web CIS. More specifically, developer collaboration and the collaboration between developers and end users is of interest.

With regard to software artifacts and software engineering, CoPs are a way to increase learning, but also to help developers and community members without technical knowledge to engage in software design and development [BCR04]. Professional end user developers [Seg07b, Seg07a] are considered to have better support and training when whey work in CoPs, developing code both for own use and the local community. Segal concludes: "agile methods appear to be a likely source of practices etc. which both support the way professional end users currently develop software and strengthen the community of practice. However, the effect of the introduction of agile practices into a professional end user developer community is still to be investigated." In investigating the use of software artifacts and the development processes used in CoPs, we build on a collaborative conceptual modeling and software development approach, based on the "evolution by collaboration" idea.

End-User Development and Participatory Design. Requirements engineering, one very important aspect in CISs has been studied in previous work conducted at RWTH Aachen University [Kla10b, Han14]. [Han14] presents the end user integration from the requirements engineering perspective, introducing the importance of EUD and participatory design in the context of open source CoPs. EUD is seen as the means for non-professional software developers to design, develop software artifacts [KMR+11, LPW06] and has been introduced in early 80's [Mar82]. The notion also refers to software maintenance, covering the entire software lifecycle from incipient stages until its deprecation. Other synonyms or related terms to EUD are mutual development, co-development and participatory design. These refer to activities in which end users are involved. However, they may be used with different focus. As an example, co-design or participatory design mostly refer to system design, and thus end user may or may not be involved in actual coding stages [HK92, CMPP08, Mac90]. More details on related approaches are presented in [Han14], which integrates a taxonomy of participatory design practices, extracted from Muller et al. [MWW92]. Fisher [Fis01, YF07] describes EUD as organizational, collaborative practice that offers insights on how people appropriate and customize software artifacts. It is also seen as a mechanism to better respond to end user requirements, as their integration brings much more domain knowledge into the software design and development processes. In our work, by EUD we do not refer to a passive integration of community users into design processes (i.e., in the meaning of participatory design through availability for workshops and/or

reviews), but rather investigate an integrating agile approach containing the software design, development, refinement and maintenance, using synchronous collaboration, conceptual modeling and model-based system engineering.

2.1.3 The ATLAS Model

Summarizing six years of CIS research, Klamma [Kla10b] describes the Architecture for Transcription, Localization and Addressing Systems (ATLAS) research methodology, which incorporates community members as stakeholders in requirements engineering and software engineering, in an attempt to scaffold community members to build system artifacts themselves, without the need to always rely on software engineers. Fig. 2.3 presents the methodology, which is rooted in the transcriptivity theory, as a basis design principle of technology [JK05].

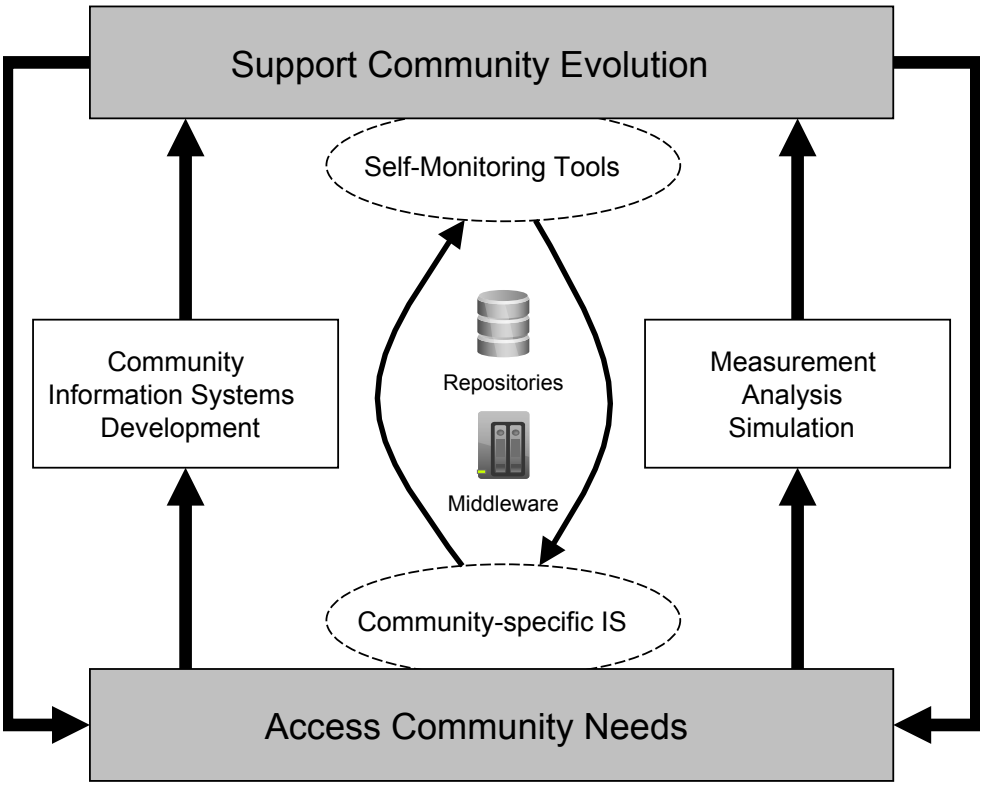


Figure 2.3: The ATLAS research methodology [Kla10b]

Jäger's theory of transcriptivity ("Transkriptivitätstheorie") [Jäg02, JJKS08] is based on three basic operations of transcription (i.e., improving readability and making sense of "pre-texts" – media artifacts), addressing (i.e., transforming of "unaddressed" information in "addressed" messages), and localization (i.e., adopting media to local practices). The theory is used in [Kla10b] to "understand digital media support for cross media community

information systems" and obtain a design oriented theory for Computer Science [JK05]. Moreover, in order to conceptualize the relations between media, cross-media extensions, and social relations between actors within a CoP, the Actor-Network Theory (ANT) [Bru99] has been used in combination with the design-oriented approach [SKJ03, KSCJ06]. This lead to a model for digital social networks in CoPs, which considers the actors and relationships between them to be the core of CISs. Therefore, the ATLAS community model [Kla10b] is a metamodel for describing the interrelationships between acting entities in a multimedia domain, as for example the Web. Fig. 2.4 shows the ATLAS community metamodel. It can be observed that the central entity is an actor with certain attributes

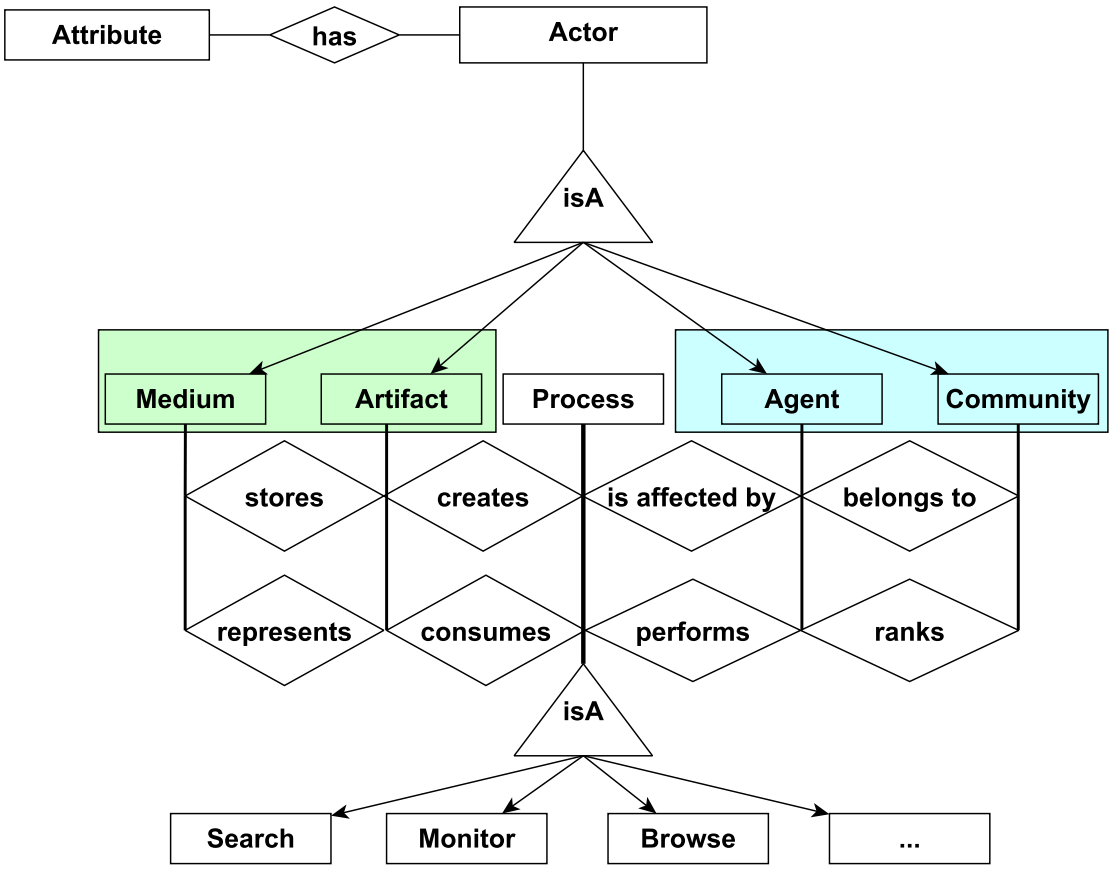


Figure 2.4: The ATLAS community metamodel [Kla10b]

that can either be a medium, an artifact, an agent or a community. As the figure suggests, there is a strong relationship between medium and artifact, as an artifact always belongs to a medium. Also agents and communities are strongly related, since communities consist of agents. The agent is either performing processes or is affected by them. Processes are basically all types of actions that can be performed on artifacts. We build on the ATLAS metamodel as a basis for our theoretical concept (cf. Fig. 6.2), seeing the "community"

entity as a CoP, with its members being agents. Using NRT collaboration and modeling abstractions, we propose to examine how can community developers and members without technical knowledge be supported in their practice and in developing customized tools. In "social participation" described by Wenger [Wen98], collaboration is involved in four main components of learning processes: meaning (experience making and knowledge refinement), practice (learning by doing), community (belonging) and identity (learning as becoming). By giving CoPs the right collaboration tools and methodologies, we want to generate a "culture of participation", an agile, co-design approach where not only developers but also other groups of users that belong to the CoP, like end users, can share their ideas and work together on constructing prototypes. These can serve as a first step towards bridging the gap of supporting more than single CoPs.

2.2 Agile Methodologies for Web-based Information Systems Engineering

2.2.1 Web-based Community Methodologies, Infrastructures and Architectures

Technologies used in CIS are based on certain architectural models, well known in the information systems and software development areas. These are structured according to available requirements, protocols and infrastructures. They also need to be determined with respect to certain constraints [DKK+13, DKN+14]:

- Federated/distributed and P2P architectures: built up of independently deployed computing nodes that are linked together by a network structure, operating collectively. They support different application types (new, third party, legacy) in a distributed environment. A major role for the information integration is played by the data exchange between the applications. Low coupling is one of the most important features, as strongly interconnected components are usually grouped into domains. Further characteristics include high flexibility, reduced complexity (for individual components but harder to maintain from an overall perspective), loosely coupled components with a high inter-component agility, no central authority (i.e., increased robustness), platform independence (multiple programming languages can be used), single components can be developed independently. P2P system design has always been mainly driven by the idea of a completely decentralized Internet architecture, without any central authority "owning" the content stored in the network.

- Service oriented architecture (SOA) [Erl05]: exposes well defined functionalities of a system as services. SOA supports important characteristics of software systems, such as loose coupling, reusability and a simple/composite implementation. However,

the usual focus is towards the support of business functions and processes across enterprises or organizations and the service orientation of existing software landscapes. A pure SOA architecture can prove difficult to implement in federated infrastructures.

- Resource oriented architecture: applies the principles of Representational State Transfer (REST) [11] and specifies how RESTful Web Services are developed. Design principles are statelessness, addressability, connectedness and the uniform interface.
- Cloud computing: enables on-demand, scalable network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services).
- Microservice architecture: The concept of microservices has emerged in the past years [New15] as a state-of-the-art development approach for component-based architectures. Microservices unify development and operations in agile teams, concentrating the design and implementation efforts on features rather than on building monoliths. With key advantages such as high reusability, modularity and scalability, the idea of independently deploying and updating single components, without the need to redeploy the whole application, has soon found adoption in both industry and academia. Furthermore, using container-based deployment as the underlying virtualization technologies (as in the case of Amazon Web Services and multiple cloud providers), microservices nowadays build the basis for cloud architectures of big software companies like Netflix or Spotify. Microservices are often seen as a step for moving towards an agile DevOps culture, bridging continuous integration and continuous delivery, due to their decoupled and independent nature.

Various scenarios and use-case implementations in multiple CIS studies [CRJ+10, CKY+09, CGKP08, CKK+15, NRK+14, NK14] have demonstrated the need to switch from single-user to collaborative, multi-user applications and to transform the community experiences in terms of frontends and classical software architectures. The Web can be seen as a programmable platform, composed from a large number of heterogeneous Web-delivered services (such as RESTful Web services, cloud-enabled APIs, SOAP, RSS). With rising popularity, RESTful services and Resource Oriented Architecture have become an often considered alternative to the classical Service Oriented Architecture [NK14]. Pautasso et al. [PZL08] analyzed loose coupling in different kinds of Web services as a fundamental concept in the REST architecture style and identified important differences between classic and RESTful Web services.

Fig. 2.5 presents a set of core non-functional requirements that are related to the various architectures presented before. Driven by the Web, cloud-based solutions and more recently microservices (same as service-oriented architectures) received much attention and much know-how emerged with respect to engineering information systems that fulfil the selected requirements. New forms of social software continued to emerge, facilitating advanced interactions (such as crowdsourcing, NRT collboration, content production, social networks)

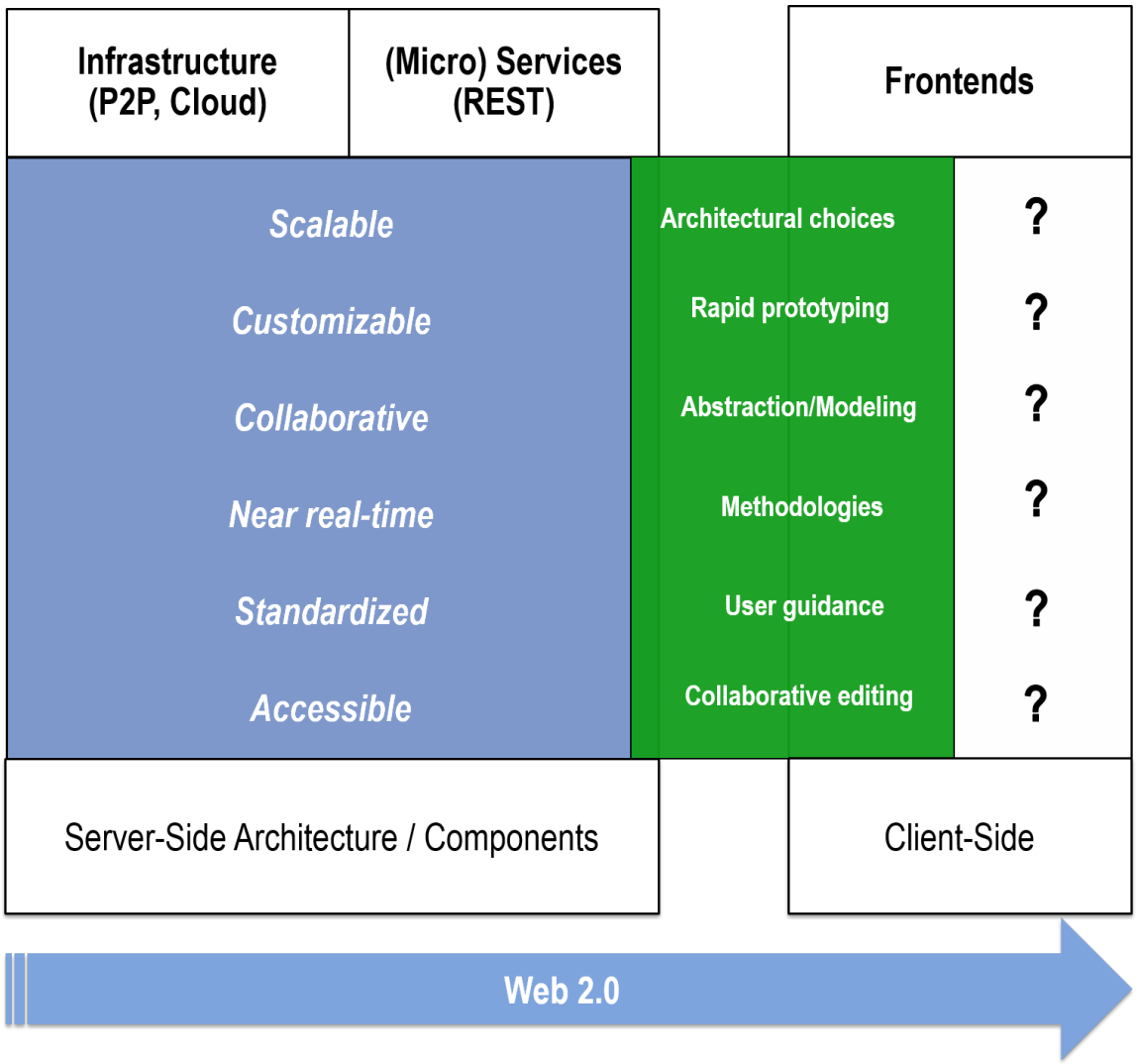


Figure 2.5: Server-side vs. client-side paradigms

and fundamentally changing the Web's role in society. Examples include microblogging, real-time Web search, monitoring and tracking, instant messaging and online A/V conferencing, social networking platforms, collaborative editing, etc. In terms of NRT collaboration adoption, many big Web industry players (Google, Facebook, Firebase, Microsoft, Mozilla, Etherpad, etc.) started to offer APIs and services in the cloud in order to support various group work, following various marketing and business models. However, due to the increased power of computing devices and the distributed communication protocols recently available (for example XMPP, WebRTC), frontends started to offer a viable alternative option to host more complex logic and operations. In this context, achieving the social Web applications requirements listed in Fig. 2.5 on the frontend and how can the interaction with microservice-architectures and the dependencies to the available infrastructures remain still

as open questions. Considering the challenges of cloud providers, such as platform dependency or data stickiness, several opportunities appear in the frontend-centric, collaborative setting:

Architectural choices The type of infrastructure or information system architecture that is best suited for designing powerful frontends

Rapid prototyping How to increase the time to market of such frontend solutions and create suitable developer support for rapid prototyping of such systems

Abstraction/Modeling Refers to the standardization and abstraction of such frontend solutions and the opportunities to model the architectures, create shared understanding among developers, use model-driven code approaches to automatically generate code for the frontends, etc.

Methodologies What methodologies can be employed for engineering such frontend-oriented information systems

Collaborative editing Since scalable and reliable cloud or client-server solutions exist, in distributed settings between Web frontends there is a gap in approaches and technology that can achieve the required performance

User guidance How can multiple users and community members be supported during their usage of information systems, with respect to awareness, recommendations, own actions and the action of other community members is still unclear in P2P settings between Web frontends

With respect to development methodologies, engineering lifecycle of CIS and the professional CoP context, the DevOpsUse (cf. Fig. 2.6) approach has been developed [DRN+15, RKKJ17], based on the ATLAS methodology. This is an extension of DevOps (a composition of the terms "development" and "operations") [Hüt12, BWZ15], an agile development practice that achieves a balance between rapid prototyping and a stable software operation through the integration between implementation and deployment tasks. Additional to DevOps, the DevOpsUse approach also considers end users into the development process. It is based on the DevOps lifecycle model, but it also acknowledges the integration of end users into codesign, open source software development and testing. The main phases and supporting tools are also depicted in Fig. 2.6. DevOpsUse targets the CoP support through the usage of social requirements engineering tools, such as the Requirements Bazaar [RBKJ13], DevOps development concepts, monitoring and awareness tools, that fulfil self reflection requirements of communities and finally contribute to their success. The methodology is also driven by open source and collaborative software development tools (for code hosting, issue tracking and continuous integration). Among the awareness and reflection tools on the success of CIS proposed by DevOpsUse, MobSOS (Mobile Community Information System Oracle for Success) [Ren16] is used by CoPs as a data-driven framework. It enables the

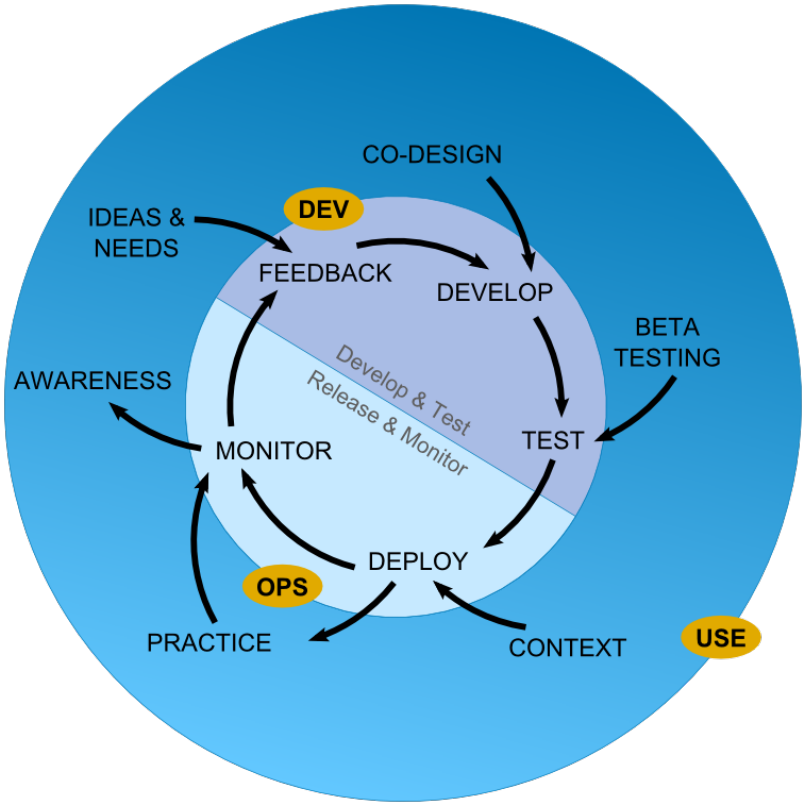


Figure 2.6: DevOpsUse methodology

management and conduction of online surveys for data collection and provides a monitoring pipeline that can collect interaction transcripts between different community agents at the communication protocol level. It also features query visualization services for community data exploration and success modeling based on CIS success metrics.

2.2.2 Model-Driven Web Engineering

Model-driven development had a first peak already during the early 1980s, with the emergence of Computer-Aided Software Engineering (CASE) [TH77] tools.

By using modeling tools, a Web application model is built, which can then later on be transformed to the actual application's source code. This modeling task usually includes the design of three orthogonal model layers, namely a frontend presentation layer, a backend data structure layer and a content and navigation model. Splitting up the model into three (or more) layers is done to achieve a so-called "separation of concerns" [SK06], where each model represents a specific part of a website, which can be altered and developed rather independently. The modeling approach for designing new Web applications bears many advantages. By building a common ground between developers and other stakeholders,

like customers or managers, it eases the understanding of how the application will look like and function, already in the design phase, without the need for everyone to understand the technical foundations of Web development. By lifting the abstraction level from code to models, the discussion about the application takes place in the problem space, not the solution space [KKK07]. This means that the focus can be put on the (abstract) design, not on how the actual implementation looks like. To define a modeling language that builds the basis for a modeling tool, a *metamodel* is used. Each element of a developed model in a specific modeling language is an instance of an element of the metamodel that describes this language. This "*meta-design*" of a modeling language, or as [ACD+14] calls it "*design for designers*", is needed to achieve another abstraction layer that makes it possible to formally specify and define a common language that is used for generating models in MDWE.

Model-driven architectures (MDA), also referred to as part of model-driven development or engineering, focuses on producing code from human designed conceptual models [KKZB08]. It can be used to tackle standardization and can separate design, architecture and development. The MDA approach is a trademark of the Object Management Group[1] that offers a standardized implementation for it [Obj03]. The major goal of MDA models are to deal with the complexity of large-scale systems by addressing interoperability, model evolution and adaption problems of Web applications [KKZB08]. Among secondary goals is also providing support in reengineering legacy systems that need to be modernized. The MDA approach mainly comprises three different types of models, also called viewpoints of the system. These are the Computation Independent Model, the Platform Independent Model and the Platform Specific Model. A Computation Independent Model only describes business concepts and requirements of the system using a Domain-Specific Language (DSL). It specifies the requirements and environment of the system. While a Platform Independent Model represents software components which do not change from one platform to another, a Platform Specific Model (PSM) is used to describe platform dependent parts [MFB+08]. By using model transformations, a PIM can be mapped to a corresponding PSM of a specific platform [MSUW02]. A PIM is a model that is independent of a specific platform, analogously a PSM is a model defined in terms of a specific platform [Fow09]. Furthermore the MDA standard of OMG includes the representation and exchange of models in various languages like UML, MOF (Meta-Object-Facility), XMI (XML Metadata Interchange) or QVT (Query/View/Transformation) [KKZB08]. Transformation and execution of models, as well as generation of documentation are a part of the standards of MDA. MDA uses a Model-to-Model (M2M) transformation to map a PIM to a specific PSM. The PSM is then used to generate the source code of the software application [MSUW02]. The MDA approach provides high flexibility and platform-independence.

[1] http://www.omg.org/mda/

Model and Code Synchronization

In the scope of MDWE, Model to Text (M2T) transformations are a special form of Model to Model (M2M) approaches, where the target model consists of textual artifacts [Mv06], in this case the source code of the generated Web application. The target model is generated based on transformation rules, defined with respect to a models' metamodel [MSUW02]. Template-based approaches are (together with visitor-based approaches) the most prominent solution for M2T transformations [CH06]. Here, text fragments consisting of static and dynamic sections are used for code generation. While dynamic sections are replaced by code depending on the parsed model, static sections represent code fragments not being altered by the content of the parsed model [ORK14]. An important aspect of M2T transformations is model synchronization. It deals with the problem that upon regeneration, changes to the source model have to be integrated into the already generated (and possible manually modified) source code. To achieve this, traces are used to identify manual source code changes during a M2T (re)transformation. In MDWE, managing traceability has evolved to one of the key challenges [ALC08]. Another challenge is the decision on the appropriate granularity of traces, as the more detailed the links are, the more error-prone they become [VBJM14, GCHH$^+$12]. In addition to model synchronization, Round-Trip Engineering (RTE) also considers changes in the source code which are propagated back into the model. Among others, formal definitions of model synchronization and RTE for M2M transformations have been proposed in [GW06] and [HLR08].

2.2.3 Related Work: Metamodels, Approaches and Systems

Related (Meta-)Models and Web-Engineering Concepts

There exist a number of MDWE (meta)models and Web-engineering concepts [MRV08], of which the most prominent are the *OOHDM*, *UWE* and the *WebML*. *OOHDM* [SR98, RS08] was developed during the early 90's and is one of the earliest attempts to cope with the changing requirements in the development of Web applications, compared to traditional software development. It is one of the earliest attempts for a structured Web modeling concept. It can be regarded as a methodology for Web applications development. Representing hyperlink-based Web systems, OOHDM follows an ad-hoc approach for Web application development rather than a structured conceptual one. "Hypermedia Applications" engineering is split into four tasks: the conceptual design, the navigation design, the abstract interface design and finally the implementation. The OOHDM approach is implemented in an editor called HyperDE [RS08]. The latest version of this editor is from 2009 and does not cope with state-of-the-art and social Web 2.0 techniques.

UWE [KK02, KK03, KKZB08, KPZM09, KK12] is a modeling language that was developed in the late 1990's as an extension to UML, a so-called UML profile. In UML-based Web Engineering (UML), the modeling of Web applications follows the principle of sep-

aration of concerns by separately modeling content, navigation, business processes and presentation [KKK07].

The UWE metamodel is an extension of the UML metamodel [KK03]. All new UWE metamodel elements are related to existing UML metamodel elements. Fig. 2.7 shows this correlation. The left side showcases three main packages of the UML metamodel, which

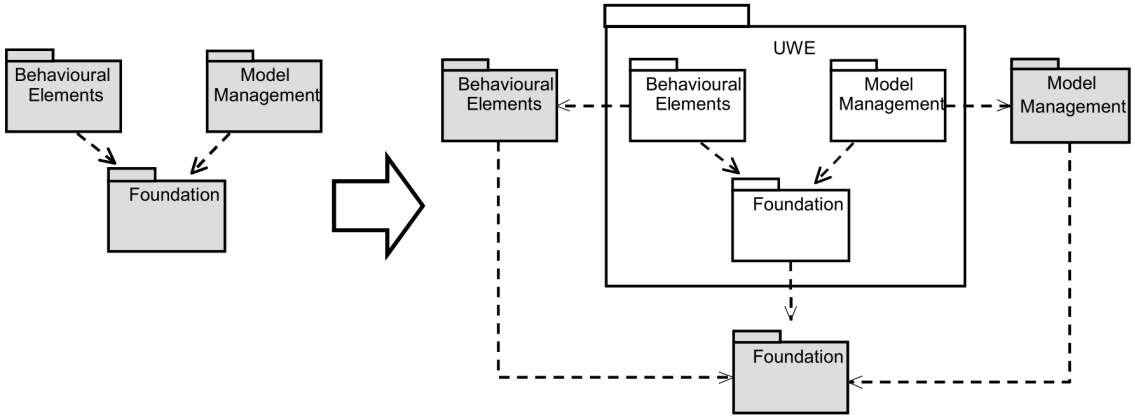


Figure 2.7: Embedding the UWE metamodel into the UML metamodel [KK03]

are the *Behavioral Elements*, the *Model Management* and the *Foundation*. The right side shows the integration of this model into the UWE metamodel. Each of the three main UML packages is represented by a main package of the UWE metamodel, representing also its connections to other represented elements. UML profiles contain three extension mechanisms: stereotypes, tagged values and constrains. The stereotypes are based on existing modeling elements and extend their semantics. Tagged values attach arbitrary tag-value pairs to any model element. UWE uses these values to represent user information in a modeling element. The Object Constraint Language (OCL) is used to define additional static semantics and constraints for a model element such as class invariants [KKZH04]. This can then be used for automated model checking. In its first versions, an UWE implementation existed as an extension of ArgoUML [Tig09], a well known open source UML modeling tool, called ArgoUWE [KKZH04]. The current UWE implementation is called MagicUWE, a plugin of the CASE tool MagicDraw [BK09]. It is based on plugins for the Eclipse platform and generates JavaServer Faces (JSF) components [KKK09].

WebML was developed in 2000 by [CFB00] as a “notation for specifying complex Web sites at the conceptual level”. Its original intent was to formulate a conceptual basis for a CASE tool suite called “Toriisoft”. Like most Web modeling languages, WebML also follows the separation of concerns idea by using an orthogonal approach splitting up the modeling task into a structural model that describes the sites content, a hypertext model that includes the composition model of the content and a navigation model that describes the topology of the links between different pages. The presentation model forms the third perspective of modeling. Moreover, WebML features a so-called personalization model, which can store

user specific content. This can be compared to the UWE's tagged values. The language also features some rudimentary user and group management, but these are rather general categorizations that do not reflect any community oriented approach.

The initial specification of the WebML language did not feature a metamodel, but a "grammar-like textual definition for specifying a structure for XML documents" [CFB00]. The metamodel was developed later by [SWK06], who argued that a metamodel builds the basis and is a must-have requirement in MDWE approaches. WebML's metamodel is based on the Meta Object Facility architecture (more information is presented in Section 2.3). The goal of this approach is achieving a common Web modeling metamodel [KK03]. The CASE suite contains a template generator that is capable of transforming the WebML specifications into Microsoft's ASP templates via XSLT, although, it seems like the "Toriisoft" project has been ended and is not available anymore. A later implementation of a CASE tool supporting the WebML language, is WebRatio [ABB+04], a commercial model-driven application development platform that supports also backend Web-service code generation for the J2EE and Microsoft's .NET platforms.

After 2013, WebML converged into Interaction Flow Modeling Language (IFML), which was also adopted as Object Management Group (OMG) standard [BF14, BMU14, ABBB15]. According to [BMU14], it specifies a platform independent user interface description through a model structured as follows: a view structure specification (definition of view containers, nesting relationships, visibility, etc.), a view content specification (such as data within a view container), events specification and event transmission specification. Fig. 2.8 shows the core concepts and notations used in IFML. *medini QVT* [GH09, ANS12], developed by IKV++, is a commercial tool that implements the declarative part of the QVT specification, i.e. the QVT Relations. Thus, it supports incremental bidirectional M2M transformations and generation of trace models during the transformation process. Models of any metamodels can be used for the transformation. It is implemented as an Eclipse plugin and provides a syntax highlighting editor, code completion and debugging facilities for the development of transformation rules. Medini QVT is available as a free version for non–commercial purpose.

UML Lab[2], developed by Yatta, is a commercial modeling suite which is integrated into the Eclipse platform. It provides a graphical UML modeling editor and template–based M2T transformations supporting reverse engineering and RTE. It supports the generation of Java and PHP source code. In addition, it provides templates for best practices and patterns, and for frameworks and technologies including JPA, JEE, the Zend Framework and CakePHP. However, NRT collaboration facilities are not provided by UML Lab.

MOFScript MOFScript[3] is a M2T transformation tool developed as an Eclipse plugin. MOFScript is a sub-project of the MOFScript language which was an initial proposal for the OMG RFP on MOF M2T transformations. As MOFScript language does not depend on

[2] `http://www.uml-lab.com/en/uml-lab/`
[3] `http://www.eclipse.org/gmt/mofscript/`

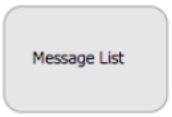
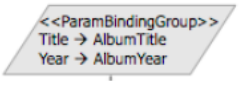
Concept	Meaning	IFML Notation	PSM Example
View Container	An element of the interface that comprises elements displaying content and supporting interaction and/or other ViewContainers.		Web page Window Pane.
View Component	An element of the interface that displays content or accepts input		An HTML list. A JavaScript image gallery. An input form.
Event	An occurrence that affects the state of the application		
Action	A piece of business logic triggered by an event		A database update. The sending of an email.
Navigation Flow	An input-output dependency. The source of the link has some output that is associated with the input of the target of the link		Sending and receiving of parameters in the HTTP request
Data Flow	Data passing between ViewComponents or Action as consequence of a previous user interaction.		
Parameter Binding Group	Set of ParameterBindings associated to an InteractionFlow (being it navigation or data flow)		

Figure 2.8: Core IFML concepts and notations [ABBB15]

any actual metamodel, it can be used for code generation of arbitrary metamodels and their instances. In addition, it supports the generation of traces as well as the synchronization between models and source code. However, it does not provide RTE facilities. Besides, the tool does neither offer NRT collaborative modeling nor coding functionalities.

During the period when many of these applications were developed, Java programming language technologies were used for Web development in many projects, especially in academia. Due to the good integration and support for Java features, the Eclipse integrated development environment was preferred to develop and showcase the various MDWE approaches (usually available as Eclipse plugins). However, being stand alone and desktop-based, Eclipse has disadvantages when dealing with collaboration and rapid prototyping in agile, synchronous environments. Even though Eclipse still continues to provide many reliable and useful extensions, agile Web development practices evolved towards the usage of multiple tools during design and implementation of Web applications. Many of these are

supported directly by modern Web browsers, the biggest players such as Google Chrome or Mozilla Firefox featuring developer tools and third party plugins that make the Web development much easier. Furthermore, for easy access and modern look-and-feel, most design and development tools also provide nowadays Web interfaces and are developed for the Web.

2.3 Collaborative Conceptual Modeling

2.3.1 Domain Independent Visual Metamodeling

In many disciplines modeling[4] is a key activity in defining high-level representations of the target domain [KOB07, DNE+15]. For any specific case which comprises a part of the real world, a conceptual model builds an abstraction by simplifying its possibly complex structure to just the key parts relevant for the outcome of the software engineering process. A conceptual model [Oli07], as well as the domain it represents consists of objects and relationships between these objects. Thus, there exists a one-to-one correspondence between the objects as part of the real world and the objects of the model [HPv05]. Furthermore, objects and relationships which share the same logical context are classified into concepts. This definition of a conceptual model assumes a particular view of the real world, i.e., the real world consists of the three mentioned components.

Conceptual modeling languages can depend on a specific domain or can be defined for a general purpose (such as UML in software engineering). For certain domains, reference models are used as generic conceptual models that generalize and formalize best practices from the respective areas. The reference models span from specific domains such as "financial accounting" or "federal enterprise architecture" to the scope of an entire industry sector, such as "higher education" [Rv07].

Models obtain meaning through creative social processes, driven by the meaning negociation of the involved stakeholders, which usually have heterogeneous perspectives [DFK02]. A further characteristic of a conceptual model is the independence of its subsequent technical implementation. On the modeling level no restrictions or constraints need to be considered regarding the final realization of the model, as in the case of a programming language.

For the definition of a model a particular modeling notation is used, determined by the modeling language, which consists of a syntax and a semantics definition. The syntax definition constitutes the elements of the language and their syntactical notation, whereas the meaning of these elements is referred to by the semantics definition. We focus on graph-based modeling languages, that is, languages whose elements are represented either as nodes or edges. There are two different user interaction paradigms for graph-based diagrams, namely free-hand editing and structured editing [Min07]. The former allows

[4] the concepts exposed here were gathered in related works [DNE+15, NRD+16, NRD+18]

the modeler to create and combine model elements without any restrictions, potentially leading to syntactically incorrect models. This is avoided in structured editing by offering only those operations that lead from one valid state to another. Obviously, while free-hand editing provides freedom, structured editing scaffolds the creation of valid models. As mentioned before, a model always builds an abstraction for a particular domain. Hence, modeling languages are mostly referred to as Domain Specific Languages (DSL) resp. Domain Specific Visual Languages (DSVL), depending on the notation of the model either in a textual or graphical manner [EJ01, FP10]. The syntax definition is distinguished into an abstract and a concrete part. The former one defines the elements of the language, i.e. the attributes of its objects and their relationships. Rules regarding the notation of the elements in textual or visual form as well as regarding combinations of objects and relationships are given by the latter one. The meaning of each entity of the model is declared by the semantics definition. The modeling language itself can also be defined using a model, called its metamodel [AK03]. The (linguistic) metamodel represents the abstract syntax of the underlying modeling language. It specifies which types of objects and relationships can be used within a model. The elements available on the modeling language constitute an instance of a metamodel element. It is also possible to define the semantics of a modeling language using a metamodel. By that, hierarchy or inheritance relationships between objects of the model which belong to the same meta-layer can be expressed. However, due to a lack of a formal definition of the semantics, ontological metamodels are not commonly used in practice compared to linguistic metamodels. A metamodel being also a model, its structure can be defined using a metamodel, i.e. a metametamodel. By iterating this expansion to the next meta layer, chains of metamodels of arbitrary length can be built. For a better exemplification of metamodel model hierarchies, we offer a well known example from the software engineering literature, namely the Meta Object Facility hierarchy [AK03] (cf. Fig. 2.1). This is closely related to the Unified Modeling Language (UML), that reuses its notation and outlines a modeling architecture with four meta layers.

Table 2.1: Meta Object Facility Hierarchy [Obj17]

M3 (MOF)	Defines a language for specifying a metamodel (Example: MOF)
M2 (UML)	Defines a language for specifying models (Example: UML)
M1 (User Models)	Defines a language that describe semantic domains (Example: model of a problem domain)
M0 (Instance Models)	Contains run-time instances of the model elements defined in a model

2.3.2 Technical Related Work

Views and related concepts have been successfully used in many research fields, including object-oriented databases (OODB) [AB91, Run92], enterprise architecture (EA) [Koz06, Zac03] and corresponding frameworks and in conceptual modeling [KLRT13, LV02, FK13, VMP14, MBd07]. There are many domain independent conceptual modeling tools which provide metamodeling support and offer metamodel-to-model transformation. Furthermore, one can observe that most tools lack the NRT functionality, are not Web-based or open source and put more emphasis on the implementation of functionality according to the different purposes of the tools like code generation in software engineering or simulation of business process models and systems.

So far, the conceptual modeling literature has emphasized little on the opportunities and challenges of real-time collaboration in the conceptual modeling process. We aim to draw attention to this aspect both from a theoretical and practical perspective. Table 2.2 depicts an overview of various research existing in the conceptual modeling (CM) area. It presents general or specific-purpose tools and frameworks that fully or partially address some key concepts of this paper such as domain-independent (meta)modeling, metamodel-to-model transformation, viewpoints and views and NRT collaboration. Moreover, the table demonstrates the novelty of our NRT collaboration approach on conceptual modeling and the importance of views in such settings.

Table 2.2: Comparison of related tools and frameworks

Tool / Framework	Type	Domain-independent	Meta modeling	Graphical view editing	Collab. viewpoint definition	Collab. view manipulation	NRT editing	Web-based
Telos [MBJK90]	CM	●	●					
ConceptBase [JGJ+95, NJJ+96]	CM	●	●	●				
MetaEdit+ [KLRT13]	CM	●	●	●	●	●		
AToM^3 [LV02]	CM	●	●					
ADOxxx [FK13]	CM	●	●					●
Sirius [VMP14]	CM	●	●	●				
CO2DE [MBd07]	CM			●	●	●		
DiaMeta [Min07]	CM	●	●					
Gallardo et al. [GBR12]	CM	●	●					
Tiger [EEHT05]	CM	●						
TWICE [SLH14]	CM	●					●	●
Abiteboul & Bonner [AB91]	OODB	●						
Multiview [Run92]	OODB	●		●				
ARIS Framework [Koz06]	EA			●				
Zachman Framework [Zac03]	EA	●		●				
Archimate [Lan13]	EA	●	●	●				

A plethora of CM tools provide metamodeling and view extensions. One of the early studies involving metamodeling and its application in information systems engineering is Telos [MBJK90]. It is a language with concepts from the knowledge representation domain,

being the first to integrate formal logic semantics. It supports the construction, query and update of structured knowledge bases. Based on Telos, the DAIDA project [JMSV92] applies the concepts to the engineering of information systems, in the form of a framework allowing requirements modeling and design and implementation specifications. ConceptBase [JR88, JGJ+95], also based on the Telos object model combines advantages from relational and object-oriented databases. It is the first implementation that can generate code from conceptual metamodel specifications automatically. It also supports collaborative work and proposes a view concept for distributed analytics [NJJ+96, NJ99].

Among further research works focusing on conceptual (meta)modeling are Diameta, the modeling framework used by Gallardo et al. [GBR12], Tiger [EEHT05] and TWICE [SLH14]. *DiaMeta* [Min07] is a platform-independent Java based framework for the generation of generic editors for visual languages, using the metamodeling approach for the specification of the diagram language. *Gallardo et al.* [GBR12] proposes a method for the development of domain-independent collaborative modeling tools. It allows the generation of a Java based collaborative graphical editor for visual languages of different domains. The outlined implementation also is based on Eclipse by using EMF's modeling architecture Ecore for metamodeling and the Eclipse Modeling Project's Graphical Modeling Framework as basis for the graphical editor. The *Tiger Project* [EEHT05] is also built upon the Eclipse project. However, it has a different approach in the specification of the visual language, providing a graphical tool to specify the language by a grammar instead of a metamodel.

MetaEdit+ [KLRT13] is a tool set to define modeling languages and generate model editors. MetaEdit+ employs a locking collaboration approach. It provides three types of editors to create, edit and view a graph. The diagram editor visualizes nodes and edges and allows graphical editing. The matrix editor represents the graph as matrix. The table editor provides a tabular overview on the objects and relationships and their attributes of the graph. A subgraph allows modelers to link an element of a model to an arbitrary number of subgraphs, describing the element in detail. Basically, a subgraph is a conceptual model describing an element of a model. This allows modelers to give an arbitrary depth for conceptual models.

AToM^3 [LV02] and *ADOxx* [FK13] are domain-independent metamodeling frameworks with focus on simulation of models. *AToM*^{3} allows to transform a model expressed in a certain formalism to an equivalent model in another formalism. ADOxx [FK13] provides a query language called AQL for the generation of views on these models. It allows the development of metamodels and the generation of a modeling tool based on this metamodel. It focuses on enterprise related functionalities like the design and simulation of business processes, but it is generally domain-independent. ADOxx provides a user interface to construct AQL-queries which should help users to create queries and make sure the syntax is valid.

Sirius [VMP14, Sir14] uses the Eclipse Modeling Framework (EMF) as basic infrastructure. It offers fully customizable viewpoints on complex models. Sirius allows to represent a model in various editors like diagram, matrix, table, or tree. Each editor is fully customizable. Modelers can define conditional styles and filters for entities based on their attributes. It

is possible to generate a subset of the available palette and define optional layers to show additional content. Users are able to develop custom layers with the Java libraries of Sirius and directly apply them to the models. Therefore, programing skills are required. Sirius lacks NRT collaboration features, but it offers many customization options which makes the framework very powerful.

CO2DE is a desktop collaborative modeling application. It provides awareness features to help users recognize edits of model elements and a chat room. CO2DE does not support metamodeling. Also, it does not automatically solve editing conflicts and uses a locking approach for enabling collaboration, which is therefore not in NRT. The philosophy is that users have to discuss about conflicts and deal with them on their own.

Enterprise Architecture (EA) frameworks are used to look at complex information systems from different point of view – such as data, function, networks, organizational, structures, schedules and strategy. The *Zachman Framework* [Zac03] is a two dimensional classification schema for descriptive representations of an organization. It is an abstract guideline which proposes perspectives on a particular system of an enterprise in different development stages. The *ARIS Framework* [Koz06] provides various model editors to build complex enterprise architectures, such as location allocation diagram, network diagram, technical resource model. All entities of these model editors are integrated into one comprehensive metamodel.

ArchiMate [Lan13] is an open EA modeling standard from The Open Group. The notion of viewpoints and views conform with the IEEE standard. ArchiMate strictly separates the content and visualization of the view. The visualization can be almost anything from simple static standard diagrams to dynamic visualizations like movies. For each viewpoint a set of modeling actions is defined which operate on the model or the content of a view, e.g. refining, abstracting or translating a concept. Furthermore altering operations of the visualization are mapped to these modeling actions. In contrast to our approach manipulation of the visualization may not update the view or model. The Open Group certified various applications that support ArchiMate models but these do not support NRT collaboration.

Finally, object-oriented databases fully support general concepts of object-oriented programming languages. One of the most popular view extensions is called MultiView [AB91], a simple and powerful tool for supporting multiple views in the Gemstone database [RKR+96]. Multiview introduced the Closed-View Generation algorithm to facilitate the definitions of viewpoints.

2.3.3 Awareness and Nudging User Actions in Collaborative Systems

As presented above, collaboration is an important aspect in various CoPs. Since Web-based collaboration involves members working together in remote settings (for example collaborative modeling), awareness and relevant suggestions based on actions are very important. This especially holds for settings with many users generating data or where the process should be supported and certain action suggested in an unintrusive manner. In

CSCW, the term "awareness" was introduced by Dourish and Bellotti [DB92] in the early 90's. The authors describe it as the "understanding of the activities of others, which provides a context for own activity". Generally, there is no clear-cut definition of awareness [Gro13]. When working in the domain of CoP, this understanding can then be used to select the next step for oneself that provides the best result for the CoP, which then again allows the group to progress. Current examples for awareness information provided by the application are the "who is online" functionality of chat software (such as Facebook, ICQ) or "changes history" (such as Google Drive). Early research did also already take awareness information into account. As an example, [HHWM92] implemented a text editor with a feature called "Edit and Read Wear" of a document, visualizing the progress made on the document in the side scroll bar. [Gro13] states that awareness in a collaborative environment, where users are locally distributed, tries to simulate the behavior of users that are in the same room together and working on the same sheet of paper. It tries to overcome problems resulting from not being able to directly see or hear what others are doing in distributed groupware. The way awareness is generated can be divided into two main categories. The first describes approaches where the collaborative system collects the information itself and provides it to the user, of which [Gro13] refers to as "technology-oriented side". This method can be used by the developer of the system to explicitly point the user at particular events and is thus the one that can easier be used to guide users. The second category contains systems that provide users with the ability to generate awareness information of which they think it might be relevant for others themselves. [Gro13] calls this one the "ethnographically-informed-side". Awareness has a great influence on the efficiency of a CoP, not only when working in a NRT collaborative environment but in all situations where collaboration is necessary [HSH+02]. The way awareness affects user behaviour is very much related to how the information is presented and received. This is ranging between very intrusive and overly guiding to totally unintrusive, only presenting itself in the background without consuming any screen space of the user's display. However, most systems try to present their information as unintrusive as possible in order not to disturb the users.

In distributed, collaborative settings, awareness information can also be used together with recommendations or user guidance, based on the actions of the collaboration participants. Kravcik and Klamma [KK11] present the concept of "unintrusive guidance" as a way to raise awareness and direct user behavior in CIS. Preserving the above mentioned principles, in such systems users should be able to follow a certain guidance, but in the same time this should be just as easy to be avoided. The idea is based on the concept of nudges, coined by [TS08], that started from the concept of "Libertarian Paternalism" [TS03]. The latter arguments that situations where steering people into directions that will "promote their welfare" exist and are needed. Its underlying statement is that humans do not always know what is best and they can be influenced towards doing the right thing. The concept of nudges brings this idea a bit further by introducing ways on how to present such steering information. A nudge is the contrary of a prohibition and offers freedom of choice. It should give people a suggestion for what to do, pushing them in the right direction but without forcing them to perform the suggested actions. An example and research opportunity is the application

of user guidance in collaborative learning environments and ways to influence the learning process [KK11]. In this domain, user guidance means suggesting selected learning resources and units to improve the learning curve. Usually, human actions and biases that can be predicted follow a certain pattern. The knowledge gained from analysing these patterns can be used to guide users through so-called uneasy choices. These choices (delayed effects, difficulty, infrequency, poor feedback and unclear impact) have to be supported by user guidance. As a solution to ease these choices, [KK11] suggest the following principles to design a user guidance system, which can easily be mapped from the learning environment to the CoP collaboration scenario:

- Default options: Present the user the most common next steps most of the users take.
- Expect errors: Design the system such that errors can be easily corrected. Especially when dealing with NRT collaboration, where every action (that can also be an error) a user performs is directly propagated to others, a system should expect mistakes and provide means to deal with them. An example message is “Are you sure?”, or markings for conflicting areas.
- Give feedback: Offer best ways to improve performance.
- Understand mappings from choice to welfare: users should know what happens when they take an advice given by the system
- Structure complex choices: This holds more for the learning process than for NRT collaboration, since the user guidance and steering a NRT collaboration system can provide usually does not deal with complex choices of the user, since it does not know the outcome of the collaboration process. This is different in a learning environment, where the desired learning outcome is already known beforehand.
- Put the right incentives on the right people: This holds also true for the NRT collaboration setting, although it is much harder to define what the right incentive for a user at a specific time is. In a learning environment, a user can be categorized by her learning status, which can be measured and thus she can get customized incentives. In NRT collaboration scenarios, this categorization to calculate which incentive should be put on which user depends very much on the application.

2.4 Near Real-Time Collaboration and Shared Editing

New forms of social software continue to emerge, facilitating advanced interactions and fundamentally changing the Web’s role in society. Social computing advances support NRT interactions such as high-end video conferencing, prominent in CSCW literature. While lacking the real-time guarantees of embedded systems, these technologies enable almost

seamless collaboration via the Web and could usher in a new era of social computing. There are many examples of NRT Web applications which have shaped the way Web users socialize, inform themselves and interact in private or business purpose. Among them one can enumerate microblogging (such as Twitter), search engines (such as Google Search), real-time monitoring and tracking, instant messaging (Skype, Whatsapp, etc.), social networking platforms (such as Facebook or Google Plus).

2.4.1 NRT Collaboration on the Web and Groupware Literature

NRT Communication on the Web Recent advances on the Web allowed the development of fast and reliable communication protocols, such as WebRTC, Websockets, XMPP over Websockets, Server-Sent Events, which were also quickly adopted by Web communities, industry and academia. Besides enabling the message propagation technology stack required for shared editing systems, such protocols can leverage the use of peer-to-peer (P2P) shared editing algorithms and offer a fast and reliable communication infrastructure for their implementations. Technologies such as HTML5 Web Workers also make it possible to run code in the background, without any influence on the performance of Web pages. Thus, P2P approaches became a viable alternative to the more traditional client-server approaches [ATS04]. This is due to the fact that each collaborator receives the updates in a N-way communication and applies them locally, without the need to wait for acknowledgment data, orders or transformed operations from a central server. In consequence, existing NRT collaboration algorithms need to adapt to new performance requirements, which is not a trivial task, especially in the case of massive collaboration, with a scaling number of users and changes [OUMI06].

Taking advantage of the improved NRT communication protocols, remote working environments, as well as the amount of group work performed in the Web browser in various personal and workplace settings have increased dramatically. The importance of handling distributed digital artifacts and ensuring data integrity in replicated settings is growing due to an increased demand for availability and performance in collaborative NRT editing systems[5]. Therefore, algorithms and multiple implementations that can ensure data consistency in such multi-user, NRT collaborative and distributed contexts resulted, as research groups and Web industry players tried to cope with these requirements. Quite popular in the Computer Supported Collaborative Work domain, shared editing mechanisms (i.e., algorithms, methods, systems) have been the focus of several research groups in the past three decades ([EG89, SE98, IN02, SPBZ11b]), but also came more recently into the attention of big Web industry players such as Google, Microsoft, etc. (as such companies developed systems like Google Drive, Google Docs, Etherpad, Wave). As the shared editing field is rather narrow and very specialized, the communities here are well connected and situated in the periphery. Their application into more general, CSCW domains and integration into other systems mostly represent the links to the core community. However, this connection is growing and

[5] In this work, we will use the terms NRT collaborative editing and shared editing interchangeably

there are still many other applications in general – not only from the CSCW domain – which can benefit from such technologies. An example for the usefulness of NRT collaboration is in both on-site professional and amateur CoPs [KNK14]. These communities are characterized by a high degree of collaborative work, mobility and integration of data coming from many members. Collaboration facilitates peer production [Ben06] and provides the ability to iterate quickly by permitting members to work in parallel, thus increasing productivity.

In general, a real-time collaborative editing system (RTCES) [EG89] is seen as a CSCW application that enables a group of individuals to work on the same document at the same time independent from their geographical location using the Internet as the communication medium. An aim of these systems is a high (local) responsiveness with respect to user interactions, in the best case being as responsive as a single-user application. Furthermore, editors should provide an unconstrained interaction to the user, i.e. no restrictions in editing a shared document with regard to parts of it that are already modified by a remote user should be perceived. Such systems usually maintain a consistent document state [SE98] on all connected sites.

A common problem faced in collaborative work and shared editing is maintaining the same representation of digital artifacts (i.e., documents) across clients, which is referred to as consistency maintenance (or consistency control) [EG89]. Early studies present various techniques for solving conflicts between such representations across collaborating sites, in collaborative editing settings. Coined during late 1980s, concurrency control includes locking (pessimistic, mutually exclusive mechanism), transactions, single active participation (token-based participation), dependency detection (timestamps for conflict detection), differential synchronization (client-server asynchronous approach) [Fra09] and three-way merge (classic versioning systems). These can be classified into two main categories: conflict prevention [XZS00] and conflict resolution [SJZ+98]. The former one comprises mechanisms to prevent the occurrence of conflicts, for example by restricting the user's access rights to just those parts of the document that are not manipulated by another user. Thus, consistency is achieved as each performed operation leads from a consistent state into the next one, i.e., there are no conflicting operations. However, systems implementing this type of collaborative editing strategy are usually not working in NRT (for example git, Subversion). The latter approach permits the free editing on any part of a shared artifact, thus the occurrence of conflicts originating from multiple users editing concurrently the same data and propagating their changes using the network (i.e., Web). Handling and solving such occurring conflicts is named conflict resolution. In contrast to transaction management systems or versioning protocols, shared editing systems have to cope with multiple clients joining and leaving a collaborative working session, NRT propagation of updates, etc. Such distributed systems challenges are recognized, the most important being entailed in Erik's Brewer "CAP theorem" [Bre00] which states the impossibility for a distributed system to simultaneously satisfy in the same time three properties, namely consistency (i.e., every read receives the most recent write or an error), availability (i.e., each request receives a (non-error) response without guarantee that it contains the most recent write) and partition

tolerance (i.e., despite an arbitrary number of messages being dropped (or delayed) due to the network, the system continues to function correctly). In systems design, the choice is in fact between consistence and availability, in case the system can fail. However, it has been shown later that the CAP problem can be solved by allowing for strong eventual consistency [SPBZ11a]. Here, Shapiro et al. define consistency as a groupware replicated system property, where regardless of the network latency or order in which shared edits are propagated and received, all shared replicas for all participating sites are identical. Based on this idea, eventual consistency can be achieved if delivery (i.e., an update executed on a correct replica eventually will be executed on all correct replicas), termination (i.e., all updates executions terminate) and convergence (i.e., correct replicas that have executed the same updates eventually reach the same state) conditions are satisfied. Finally, strong consistency is fulfilled where for all sites that have received the same (unordered) set of operations, shared replicas are identical.

Usually, NRTCES are based on a centralized or replicated (decentralized) architecture [GM94]. By centralized we refer to client/server architectures, where all connected clients transmit their operations to a central server, which stores the general document state and is responsible for solving eventual conflicts and propagating the corrected replicas or operations back to the clients. In contrast to this mechanism, the P2P architectures [SJYZ97] moves the conflict resolution to the client and does not require a central processing component. In this case each node directly broadcasts an operation to all the other nodes, whereof each single one computes the conflict resolution itself. The client/server architecture can save computation effort compared to the P2P approach as the conflict resolution is just computed once and not per node. Nevertheless, the typical disadvantages of a this architecture also hold in this case: the server constitutes a single-point-of-failure and may also be a bottleneck with regard to the computation time and the network delay. Even more, they do not always scale well [OMUI06], depending on the use-case they need to support. In edge or fringe computing, decentralized settings, such solutions cannot be employed. Algorithms that need preprocessing on the server side cannot be used in such P2P settings. Algorithms that do support broadcast P2P message propagation still rely on propagating a state vector (for example COT [SS06]) with each message.

The most prominent mechanisms for optimistic concurrency control are Operational Transformation (OT) [EG89, SE98, SJZ+98, SXS+06, SXA14] and Conflict-Free Replicated Data Types (CDRTs) [OUMI06, PMSL09, WUM09, RJKL11]. The OT [SJZ+98] approaches define how to handle conflicting operations for the collaborative editing of text documents. To avoid the occurrence of inconsistent states caused by the application of conflicting operations, algorithms adjust the parameters of an operation dependent on the previous applied conflicting operations.

In contrast, CRDT algorithms offer optimistic concurrency control by defining commutative operations that do not conflict with each other [SPBZ11b, SPBZ11a]. Hence, they try to avoid conflicts through the design of certain data types. In comparison to the conflict resolution the conflict prevention approach causes less costs with regard to the computation

and communication effort. On the downside, the user restrictions minimize the usability of the editor. For the detection of causal dependencies between two operations, i.e. whether one operation depends on the result of a second operation respectively to determine the order of two operations, a time synchronization mechanism has to exist, such as by attaching a vector clock time stamp to each transmitted operation. To ensure a high responsiveness of the editor to the user and thereby increase its usability, local changes are directly applied to the local document state, providing the user with direct feedback to his input. In a second step, the change is propagated to the connected sites resp. occurring conflicts are resolved.

2.4.2 Related Work: Algorithms and Web-based Systems

Operational Transformation

OT systems distinguish between *control algorithms* and *transformation functions* [SXA14]. The former determine the concurrent operations to be transformed against other operations according to concurrency/context relations. Capable of supporting concurrency control, they provide the time-space complexity of an OT system. Examples of OT control algorithms are dOPT [SE98], Google Wave OT [DA10], GOT [SE98], GOTO [SJZ+98], COT [SS06], TIBOT [LDS04], SOCT [SC09]. The transformation functions determine how to transform a pair of operations according to their type, position and other parameters. For the detection of causal dependencies between two operations (i.e., whether one operation depends on the result of a second operation), a time synchronization mechanism usually exists. This can be done for example by attaching a vector clock or timestamp to each transmitted operation. These systems also need certain properties and conditions, with the role of dividing responsibility between the control algorithms and the transformation functions. The properties ensure the correctness of OT systems and can be reflected on either the algorithm or transformation function level. Algorithms are checked against properties using OT puzzles, a main driver in achieving OT correctness [SXN17, XS16]. These are known scenarios in which certain transformation properties and conditions may be violated and the OT system may produce incorrect results. An overview of OT control algorithms, their verification and available state-of-the-art systems are given in [SXA14, XSL14, LXZS14, KGK13].

A very simple example of how OT works is depicted in Fig. 2.9. Two users situated at two different sites edit together a text document. At first, the text contains only two characters, namely “ab”. At the first site, the user inserts the character “x” at position 0 and concurrently, at site 2 the user deletes the character “b” from position 1. According to OT control algorithms, local operations will be executed immediately, as-is. Before the operation broadcast, the local copy will be “xab” at site 1 and “a” at site 2. At each site, remote operations will be first transformed and only after applied to the document. O_2' is the transformed O_2 operation at site 1, which results into $DEL(2,'' b'')$, making sure that character “b” is deleted and not “a” which is situated at this time point at position “1”. After

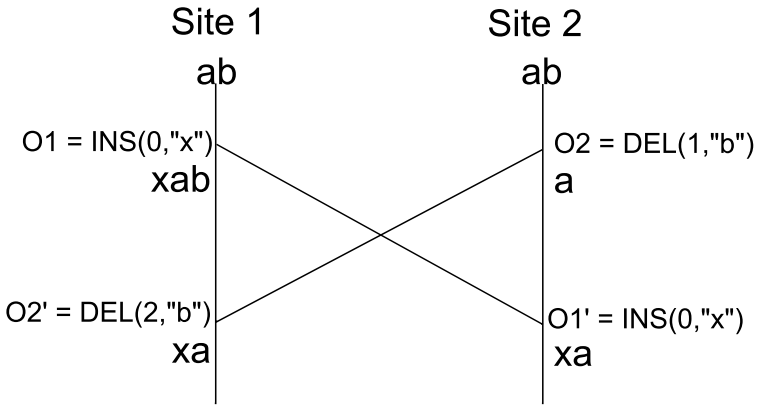


Figure 2.9: Simple OT mechanism example [Sun12]

transforming O_1 into O'_1, the insertion position of character x does not change and the O'_1 is applied. Both sites will have at the end the same text, namely "xa".

The term intention preservation is often used to describe the behaviour that the execution effect of a users operation on a local site has the same effect on remote sites. E.g. the deletion of a character, will not result in the deletion of another character.

With few exceptions, available OT algorithms are designed for client-server architectures. Most OT implementations [KGK13] can maintain consistency for digital artifacts referable via a linear address space (although they can be reflected as non-linear relationships at the UI level) or specific ones, such as tree-like [Ger06, DSL02, IN02, IN03, IN08]. The objects of a text document, i.e., its characters, have a linear relationship and thus the document can be represented as an array of characters. Operations can refer to the manipulated characters by their index. Applications based on complex models must therefore map the underlying data to the data structure that is supported by the used collaboration framework. However, this process is time consuming, application-specific, and often hard to achieve. In other cases of non-linear models, special UI level constructs are needed due to the complexity of relationships between the objects of a graphical document, that cannot be represented using a single addressing space such as an array. For these kinds of documents special conflict resolution algorithms have been developed. [FAGS11] introduces a graphical OT algorithm by extending the OT concept with regard to the requirements of a graphical editor. Approaches that are designed to support tree-like data structures require users to send a path-vector (in the size of the path from the root-element to the changed element) with each operation. The *treeOPT approach* [IN02, IN03, IN08] is a general solution to support collaboration on tree-like data structures and can be used with any OT algorithm. It can be leveraged to support collaboration on XML, since this can be internally represented as a tree.

The *Google OT* approach for Google Wave/Google Docs [XSL14] is based on the *Jupiter* approach [NCDL95]. In both cases the server has to preprocess operations before one

is executed and propagated to all clients. There is no direct communication between clients. Similarly, the client also preprocesses received operations before it executes them. Since the client has its own history buffer, it can use the same preprocessing algorithm as the server. *Generalized OT* (GOT) [SJZ+98], *GOT Optimized* (GOTO) [SE98] and *AnyUndo* [Sun02] are algorithms that do not depend on a server that preprocesses operations. The AnyUndo OT algorithm extends GOTO with undo capabilities, while reusing its transformation algorithm. The space complexity of this algorithm is nearly as good as the Jupiter approach, but its time complexity can be quadratic. Moreover, it has been shown that the GOTO and AnyUndo approaches work if there are transformation functions that fulfill CP1 and CP2 properties [RNRG96]. Sun et al. [SXA14] describe these transformation properties—correctness criteria used in OT literature, needed to achieve convergence (CP1, CP2). Violations of these properties result in inconsistent replicas of a shared document. Similar to GOTO/AnyUndo, the *Context-based OT* (COT) [SS06] algorithm does not need server-side preprocessing. Furthermore, the algorithm enables direct communication between the clients. The idea behind COT is to save every received operation "as is" in a document state and preprocess it afterwards given the operation context. COT has a simplified design, as only one transformation function must be defined. The downside of this approach is that a state vector must be transmitted with every operation, where the size of the state vector is proportional to the number of users in a collaborative session. The *Time Interval Based Operational Transformation* (TIBOT) approach sends operations to all other clients in an N-way communication model. However, each client is only allowed to send the operations at a certain time interval [LDS04]. This significantly reduces complexity, for the price that updates may take longer to be processed. Similarly, the successor approach TIBOT2 maintains a distributed total ordering schema on the execution sequence of operations, allowing N-way communication between clients. Moreover, it avoids both CP1 and CP2 [XSL14] via a more efficient remote processing approach.

OT approaches still have several drawbacks, such as scaling problems in peer-to-peer and cloud networks with dynamic (join/leave) user behavior and a high complexity in solving convergence [ANIO+11]. Moreover, the complexity was a reason for slowing down the spread of collaborative applications on the Web, which flourished with the uprise of Google and cloud-based systems. Moreover, there are few open source implementations that work directly in Web browsers, can be used P2P and offer a developer-friendly, easy and intuitive way for rapidly enabling NRT collaboration on custom data types.

OT Web-based systems are probably the most popular applications for collaborative editing. Apache (formerly Google) Wave protocol is an excellent example of XMPP-based communication and collaboration platform for concurrently editable structured documents and real-time sharing between multiple participants. Finally, the Collaborative Editing Framework for XML (CEFX) enables lightweight concurrent real-time editing of XML files using operational transformation algorithms. Since XML has a generic and extendable nature, different kind of information can be stored such as graphic files (SVG) [Ger06] .

Conflict-free Replicated Data Types

While trying to overcome the scalability and complexity challenges in distributed environments of OT-based systems, a new family of algorithms known as CRDTs have been developed in the past decades [OUMI06, AMOI13, PMSL09, RJKL11]. The term CRDT was coined by Preguica, Shapiro et al. [PMSL09, SPBZ11b, ANIO$^+$11], but earlier works performed for text editing system were already adhering to the core ideas of CRDTs, trying to improve existing OT-based algorithms (such as the WOOT [OUMI06] and Logoot [WUM09] algorithms). CRDTs propose to ensure consistency on data types satisfying a set of mathematical properties, that guarantee strong eventual consistency. These data types can be replicated across multiple servers or can be used also on the client side. CRDTs can be used in large distributed systems such as replicated databases, however they have emerged trying to improve OT solutions for text editing, in P2P environments [OUMI06]. These have a small document update size and do not rely on vector clocks (for example the WooT approach [OUMI06]), and can therefore suit the new communication protocols and practices on the Web.

Eventual consistency allows for the execution of updates without synchronization of each replica. Concurrent updates causing conflicts may need reconciliation or a roll-back of executed updates [SPBZ11b]. However, strong eventual consistency (SEC) does not need any reconciliation strategy. In this case, every update can be applied directly on the data type [SPBZ11b]. Starting from the SEC notion, the mathematical properties of CRDTs are based on a monotonic semilattice. This denotes a partial order relation, with a least upper bound. A partial order relation on a given set is reflexive, antisymmetric and transitive.

Due to these properties, CRDTs enable the creation of specific data types, such as counters, lists, ordered trees, etc. These are replicated over several sites. The operations applied to the data types have to be commutative, such that contrary to OT functions no merge algorithms or integration steps are needed. The operations are associative, commutative and idempotent [SPBZ11b]. This particular feature is the reason why CRDTs belong to the conflict prevention algorithm family and do not need concurrency control to ensure consistency (i.e., conflict resolution).

Due to their properties, CRDTs are scalable in terms of number of users and guarantee a high availability, also in P2P settings. In a distributed system with SEC the replicas make local updates of data and then propagate them to the other replicas. All correct replicas will have equivalent states after they executed the same updates, without any reconciliation. This property is called strong convergence [SPBZ11b]. SEC has a deterministic outcome for any kind of conflicts. The absence of a need for consensus improves the performance of the systems, increasing their availability. In consequence, SEC-based applications are fast and predictable.

CRDTs can be state-based or operation-based [SPBZ11b]. State-based CRDTs send their entire state to other replicas in order to achieve consistency. A $merge$ operation is executed upon receiving an update. Such CRDTs require a partial order of operations, with the

updates increasing the replica states monotonically and merge functions computing a least upper bound. The operation-based approaches only send individual operations to remote clients and do not require the propagation of other information in order to ensure consistency.

Examples of CRDT data types are [SPBZ11a]:

Counter A replicated integer, with defined "increment" and "decrement" operations. An example is an increment-only counter, which can only grow in value, such as the number of computer mouse clicks or the number of likes on a social software platform.

Register A data store, with defined "assign" and "value" operations. An example is a last-writer-wins register (the merge is based on a total order of timestamps, assigned to each operation).

Set A shared data set, with defined "add" and "remove" operations. An example is a grow-only set which implements the set union operation for adding elements.

Graph Is realized as extension of the Set data type, namely a tuple of edges and vertices sets, with defined "addVertex", "removeVertex", "addEdge" and "removeEdge" operations.

CRDT Algorithms for Shared Editing

Without OT (WooT) [OUMI06] is an approach built for consistency preservation in P2P systems. It uses a monotonic linearization function to ensure convergence as a solution for topological sort. WooT supports linear structures, lists, block of texts and ordered trees data types [OUMI06]. Moreover, it makes use of an unique identifier for each collaborating site and one for each operation, a value for the effect of an operation and the identifiers of previous and next operations for each given "character value" of an insert operation. WooT uses tombstones (i.e., no elements are deleted, they are only marked for deletion). Each site maintains besides the identifier a logical clock, a sequence for character values and a structure for pending operations. WooT is independent from the order of reception of operations.

WooTO and WooTH [ANIO+11] improve the WooT algorithm in terms of performance, via a linked list and a hash table for optimizing retrieval, update and insertion of operations. They were inspired by the replicated growing array (RGA) approach [RJKL11, ANIO+11], which introduced update operations in addition to the classic insertions and deletions.

Logoot [WUM09] provides a totally-ordered set of elements and was developed for linear data, extended with an undo mechanism based on a PN-Counter [WUM09, SPBZ11b]. It does not require tombstones, which makes the algorithm more efficient compared to similar mechanisms such as WooT.

The Replicated Growable Array (RGA) [RJKL11] is an optimized CRDT algorithm that builds upon ordered lists (i.e., a linked list). It supports insert, delete and update operations with the possibility to change an element without an impact for the document

size [ANIO+11]. Operations are identified with the help of indexes, which are stored as integers. For remote operations, a hash table is use in order to compare local indexes with remote ones. A $s4vector$ is introduced as the unique index for the RGA [RJKL11]. The integer values of $s4vector$ specify the session, the unique identifier of the site, the sum of the vector clock and an identifier to purge tombstones [RJKL11]. Every operation results in a unique $s4vector$. Based on the $s4vector$ construct, the position of new elements in the linked list can be determined. Each new element is appended before the element with the next older $s4vector$ [ANIO+11].

Table 2.3: CRDT systems and libraries

Tool / Framework	Algorithm	License	Data types	Doc
woot.js [OMUI06, Dal17]	Woot	Apache v.2.0	text	●
elm-logoot [WUM09]	Logoot-Undo	-	List, text	●
peeredit [Ore17]	RGA	-	text	-
js-crdt [SPBZ11b, Tar17]	Own	MIT	Map, Set,Sequence	-
Swarm [Gri17]	Own	MIT	CRDT, RPC	●
Akka.net [akk17]	Own	Apache v.2.0	CRDT	●
Riak [Aca17]	Own	Apache v.2.0	CRDT, Hyperlogs	●

Table 2.3 presents several available CRDT implementations. Overall, the Web libraries implementing CRDTs have a rather reduced adoption with respect to the number of visible users. This is the example of Javascript libraries implementing shared editing CRDT approaches such as Woot or Logoot. They also do not have a good documentation, being more prototypes and demonstrative realizations of CRDT concepts. On the other hand, more mature frameworks such as Swarm, Akka.net or Riak are used as shared databases or for server-to-server reliable communication in highly distributed environments.

Woot.js [Dal17] implements the Woot algorithm, using the Scala and JavaScript programming languages. The project contains a client and a server implementation. It uses Web sockets for propagating edit updates. The server can store the state and internal data representation of each document. The library can be used for shared text editing.

elm-logoot is a small project that implements the Logoot-Undo algorithm proposed by Weiss et al. [WUM09] for P2P shared text editing. Peeredit [Ore17] is an implementation of the RGA CRDT algorithm. It presents a Web applications with a client and server components for online shared text editing. Being a proof of concept, it lacks documentation and adoption. Finally, js-crdt [Tar17] is better known in the open source developers community (i.e., having over 500 stars on GitHub) than the frameworks mentioned above. It also provides support for several data types except text and lists, including Map, Set and Sequence. Based upon the work of Shapiro et al. [SPBZ11b], the library provides an own implementation of the CRDT approaches. Its documentation only instructs on how to use the JavaScript code.

Swarm [Gri17] was created as part of a collaborative editor and evolved into a data synchronization middleware, with both NRT and offline modes. In its current version, it is

considered to be a "sync-centric isomorphic database", i.e., can run on both client and server side applications and synchronize the shared data between them. Swarm implements an operation-based CRDTs and supports the known CRDT data types, such as Counter, Set, Graph, etc. However, it can also support remote procedure calls (to avoid global operation propagation) and it also implements a crypto currency [Gri17]. Akka.net [akk17] is a complex project, a collection of open source libraries for designing scalable, resilient distributed systems. Among its components, CRDT data types for synchronization are also implemented under distributed data. These are flags, counters, sets, dictionary and registers. The implementation still relies on tombstones, but the associated high memory load is reduced by replicator, which removes data associated with no longer existing nodes (i.e., pruning). Finally, Riak [Aca17] is a distributed, decentralized database system. Riak's data types implementation is based on Shapiro et al. [SPBZ11b] and include counter, flag, hyperLogLogs, map, register and set. Client libraries are implemented for most common programming languages like NodeJS, Python, Ruby, Java or PHP. Riak Data Types are operation-based and allow operations such as removing a register from a map or enabling a flag that has already been disabled [Aca17]. It is probably the most mature, well documented and widely adopted implementation of CRDT.

Overall, shared text editing solutions are well covered both in literature and practice. Many OT-based solutions are currently existing in commercial systems implementing synchronous text editors (big IT companies such as Google, Apple, Microsoft, Dropbox, etc.). However, CRDTs are gaining interest and adoption as well. They are suitable for building complex data types. Also, they are designed to scale with the number of users in distributed settings and have already been adopted by industry players such as Facebook, bet365 and TomTom.

2.5 Development Testbeds

In this section, we explain the various testbeds used in this work. We also present necessary background information about various projects and prototypes developed at the Chair for Information Systems and Databases, RWTH Aachen University, on which we build upon or relate to in the dissertation.

2.5.1 NRT Collaboration and Multimedia Metadata for Professional Communities

CIS for Multimedia Metadata

Based on the transcriptivity theory and the media-oriented CIS approach, previous research deals with supporting formal and informal learning in online professional CoPs, in domains such as cultural heritage. Kovachev et al. [KRKC10, KNK14, KK14] developed a cloud platform that supports professional CoPs to use multimedia information systems, with focus

on mobile devices and –based on parts of this work – for the Web. In the context of the Virtual Campfire (VC) project [CKJ10], advanced applications for CoPs were developed. Among its main contributions, a framework offering services for mobile (mobile) multimedia management, multimedia semantics, metadata, context management and uncertainty management was offered. Some of the middleware technical artifacts acting as platforms and CoP application development enablers within Virtual Campfire were the Lightweight Application Server (LAS) [SKJR06], for simple user and community management and CAELUS [Kov14], an information systems architecture for mobile multimedia services using cloud principles.

Using cloud-enabled services such as transcoding, metadata, streaming, persistence, etc., [KK14] proposes a framework for improving professional CoPs practice through offloading cloud resources to the edge, community-specific devices, in a so-called fog computing paradigm. Examples of such professional CoPs are expert groups (such as architects, historians), working in the cultural heritage domain, which have to document historical artifacts/sites. Another example are communities of experienced construction workers, which need to document their knowledge and know-how for various purposes (for example informal learning for novice workers) [KNK14]. Among the testbeds used here, prosumer communities are supported to increase peer production in on-site CoPs by means of multimedia metadata. Various projects try to achieve this, combining textual annotations with augmented reality and NRT collaboration for mobile and Web-based systems. Prosumers are users which "actively participate in the creation and modification of products they consume" [TW08]. An example of the prosumer paradigm expansion is YouTube, where users can share their own videos in certain communities or simply on the Web. Such decentralized, self-driven community processes, in which members are creating, sharing and promoting content are named peer production [Wil08, Ben06, Sch12]. In these examples, the architects and the experienced construction workers are prosumers that contribute to the shared knowledge of their CoPs.

SeViAnno [RCLK10, CRJ$^+$10] is a single-user Web application that uses the MPEG-7 standard connected with Web Services for achieving semantic annotation of videos [DGL$^+$11], on the Web. As shown in Fig. 2.10, annotations can be added using a map or using the dedicated form from the annotate widget. After a video has been annotated, users can navigate through it using the available semantic annotations or the video timeline. The semantic concepts are *agents*, *places*, *objects*, *concepts* or *events* can be linked to specific parts of a video using the MPEG-7 format. The metadata services enable also video or annotation search. SeViAnno was reengineered in order to accommodate professional communities. SeViAnno 2.0 [NK14, NRK$^+$14] was implemented as a widget-based Web application and it can be used both by single users (in case there is only one member logged in into a widget space) or by communities, when several users work in the same space. Another major goal of SeViAnno 2.0 was to use a tier-based Web architecture, design reusable metadata services and allow the distribution of the Web interface between multiple devices. The correspondence between user interface and Web Services has a major

Figure 2.10: SeViAnno [CRJ+10] and DireWolf[KRN+14]

impact on the development of new and different clients with different set of functionalities. SeViAnno 2.0 is presented in Fig. 2.11.

The application is composed from six widgets:

- *Login widget* authenticates the user and ensures that he is allowed to access the corresponding Web services
- *Upload Video widget*, where multiple videos can be uploaded simultaneously using the cloud environment. The uploaded videos are transcoded using the corresponding Web services; they can also be streamed if such features are necessary. All the videos are tagged with their corresponding file name upon upload
- *Video List widget* contains a list with available videos, which have been uploaded and already contain metadata (i.e., at least the title element from the MPEG-7 description is not empty). The videos contain information about the video title and the disseminator user
- *Annotate widget* contains the plain text metadata representation: the video title, description and tags (plain text keywords separated by comma). A drop-down element

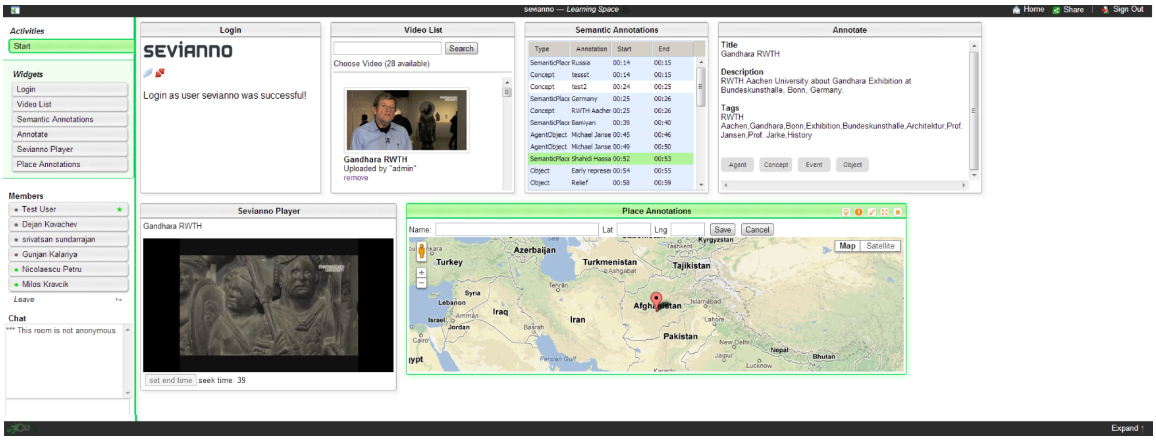


Figure 2.11: SeViAnno 2.0 snapshot

allow users to specify additional annotation types, in our case agents, concepts, events or objects. The types are mapped to the MPEG-7 multimedia content specification

- *Place Annotations widget* allows users to specify place annotation types (by specifying latitude, longitude and altitude or by navigating on the map). The widget can also display existing place annotations on the map

- *Semantic Annotations widget* displays a list with all the existing semantic annotations and their type

- *Player widget* contains the current video which has been selected from the Video List widget

[KK14] talks about the increased efficiency and costs reduction via deployment of component based clients for different domains and tasks, by following a Web service engineering approach. Moreover, it is argued that the decoupling of Web applications into multiple tiers facilitates the creation of new clients but also enables better collaboration means over the Web. The distribution has been achieved using a dedicated framework, named DireWolf [KRNK13, KRN+14], implemented on top of ROLE SDK. Using SeViAnno 2.0 with DireWolf, users can play the video in full screen on one device and use additional devices to annotate it. Moreover, they can use device-specific features for each of the UI elements, for example a multi-touch display on a smartphone for interacting with a digital map. The flexible information system architecture and powerful extensions such as collaboration and interface distribution make SeViAnno 2.0 a user-friendly video annotation tool, customizable to various scenarios/domains. Finally, the application reengineering process offered many valuable insights on developing CIS for professional, on-site CoP settings.

Widget-based Testbed: The ROLE SandBox

The ROLE SandBox is a cloud-based deployment of the ROLE SDK [GVD+11], which offers a software platform in the form of a widget-based Personal Learning Environment (PLE). The central actor here is a widget [KRN+14], a small self-contained component that fulfills a specific functionality. Widgets have limited display sizes and can be embedded in a variety of Web applications. They can therefore serve as frontends to social software, supporting CoP activities (such as microlearning, content search, collaborative editing, information display, vocabulary training, etc.). Widgets can be repositioned in the Web browser, providing a high customization to users. They can be used both for single and collaborative usage. Furthermore, they can be distributed across several devices to achieve a multi-device personal computing environment [KRN+14] and compose information dashboards or more complex applications. Due to their nature, Web widgets can provide a good solution for interacting with data as they constitute frontends for complex Web Services. However, in our studies we concentrate on building complex and flexible application mashups based on widgets. Finally, the ROLE SDK platform implements services for user management, authentication based on the OpenID Connect standard a collaborative widget space management with NRT enabled features based on the XMPP [SA09] and WebRTC protocols [BBJN13, KRN+14].

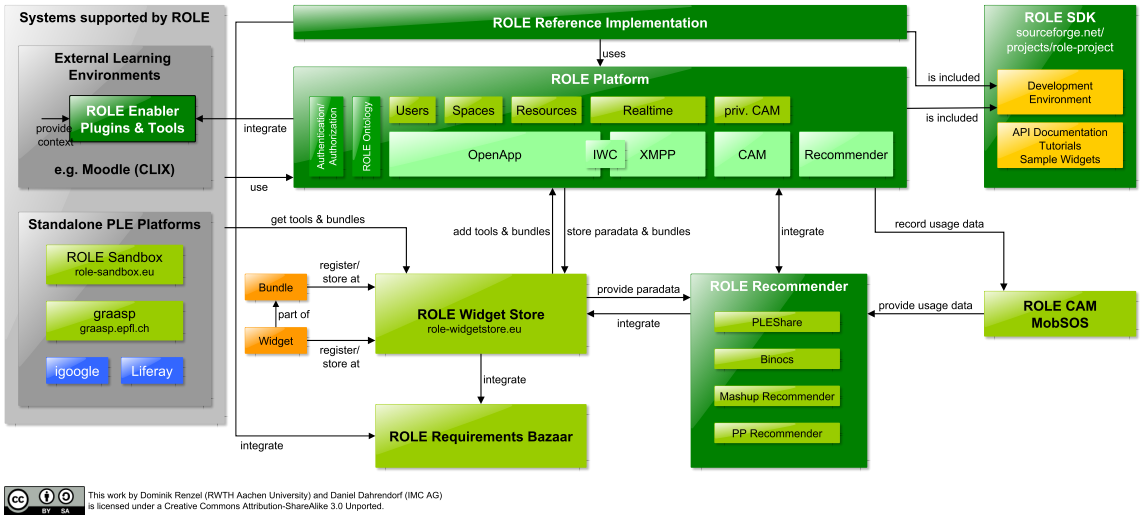


Figure 2.12: The ROLE Sandbox architecture [IPN+13, Ren16]

Fig. 2.12 showcases the ROLE SDK ecosystem of services and third-party dependencies. The spaces can be used for personal learning or as a community environment for collaborative practices or application usage. These can be tailored by using widgets or widget bundles, focusing on learning activities. The recommender component, part of the ROLE Widget store can recommend widgets and widget bundles, as well as users, resources and content,

based on domains of interest. For social requirements elicitation and community monitoring and success analysis the ROLE SDK ecosystem can use the Requirements Bazaar [RBKJ13] and MobSOS [Ren16] respectively.

In the context of our work, we have used the ROLE SDK framework as a community-enabling platform, making use of its strong community focus and the embedded user management and NRT communication facilities. Furthermore, the platform was extended – based on empirical studies for determining widget development methodologies – with state-of-the-art authentication mechanisms, and newer means for NRT collaboration. The work uses the platform both as underlying technology for the developed prototypes and the various components integrated into our MDWE approach.

Community-based Storytelling

Also part of the Virtual Campfire scenario, interdisciplinary collaboration within professional CoPs between domain experts (e.g. using multimedia, where they can be in turn amateurs or experts) was studied by means of storytelling. Collaborative multimedia tagging was applied again in cultural heritage CoPs, investigating how can data uncertainty during collaboration be reduced and how can community practices be improved by means of non-linear, community-based storytelling [Han14, Cao12]. [Cao12] identifies the collaborative story creation as a useful practice in community knowledge management which scaffold the mutual engagement, joint enterprise and shared repertoire of communities. YouTell [CKJ11], a Web-based environment for non-linear storytelling was utilized as a tool for showcasing these features using community users. Non-linear stories follow a plot that may be composed of different paths, depending on the preferences of story consumers. YouTell uses the "Movement Oriented Design" (MOD) [Sha07], an approach for non-linear storytelling. Each story is considered to be a collection of story units. Each such unit has three components called begin, middle and end that engage the consumers, carry the main message and terminate the story or link to another story unit. During the story consumption, members have the possibility to annotate media files with tags in the shape of speech bubbles. YouTell was developed as a social, collaborative application, but the collaboration could only take place asynchronously. Community users did not have the means to join a collaboration session to author or edit stories in NRT.

2.5.2 Informal Learning for Professional Communities Testbeds: Learning Layers Infrastructure

The Learning Layers project focused on developing a federated infrastructure to support and scale workplace learning in informal settings. Targeting two domains, healthcare and construction, the project built technologies for communities and clusters of small and medium enterprises (SMEs). The main results were an infrastructure for the development

and deployment of services and tools in a secure and private way, i.e., the Layers Box and the DevOpsUse methodology presented in Section 2.2. The Layers Box, presented in Fig. 2.13 represents an open source, on-premise hosted cloud-ready solution that hosts community services that can be easily deployed in CoP settings. The Layers Box is prepackaged with user management functions powered by OpenID Connect and OpenLDAP. Using a state-of-the-art containerization system, it can be deployed on various machines ranging from Mac Mini sized computers up to any cloud provider. The figure presents also a

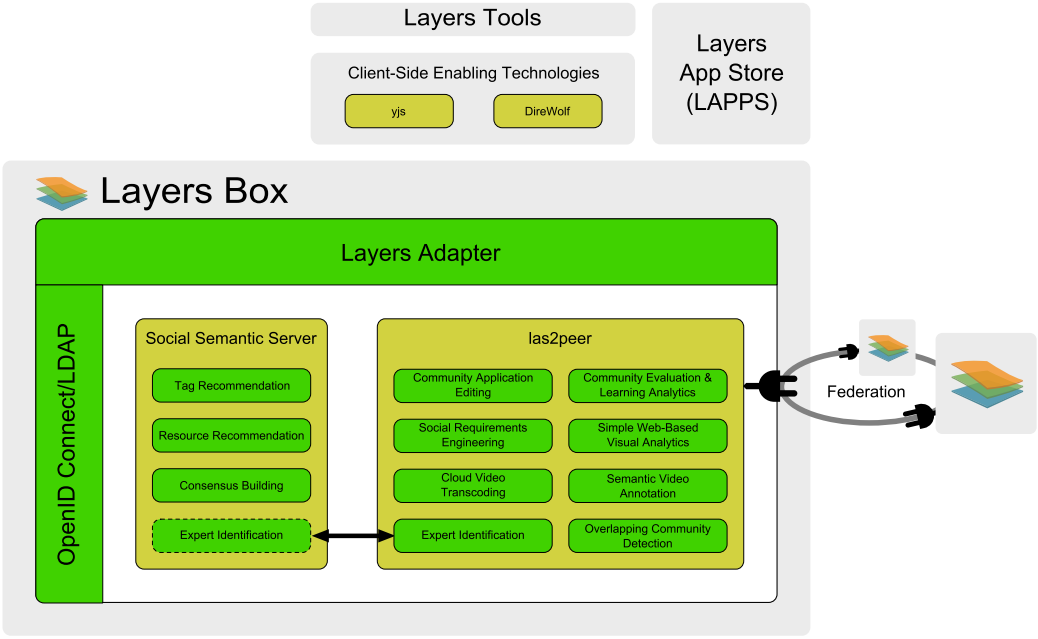


Figure 2.13: The Learning Layers infrastructure [KKd+16]

set of microservices currently hosted by the Layers Box, involving semantic metadata, recommender and expert identification systems but also other multimedia, evaluation or learning analytics components. Here, we have chosen microservices as technical framework, capable of fulfilling the needs of heterogeneous communities situated in the long-tail. The microservices, such as the ones for semantic video annotation, application editing, transcoding, etc., are developed using the las2peer framework [KRdJ16]. las2peer is a distributed and secure framework that provides a P2P mechanism and RESTful interfaces to hosted services. las2peer also provides a shared storage mechanism in the P2P network of services, providing content in a secure and scalable way, being designed for SMEs and clusters thereof, serving as foundation for the federation of Layers Boxes.

2.5.3 Learning Designers Professional Community Testbed: IMS LD Authoring

IMS Learning Design – Specification and Tools

IMS Learning Design (LD) [IMS03][6] is a specification used to formally describe the learning design of a unit of learning [HMTK04, KO04], which we refer to as the *learning design model*. A learning design model is a description of any teaching-learning process and can, for instance, be the model of a course, a seminar unit, or a self-study unit. A learning design model can be instantiated over and over at run-time in different virtual learning environments that can interpret and run IMS LD models.

Design-time vs. run-time. IMS LD differentiates between the design-time, when the learning design models are being authored, and the run-time, when these models are instantiated and executed. This enables the transfer of designs between different learning design authoring tools and the reuse of designs and materials in any IMS LD compliant run-time engines [WBP+05] such as CopperCore. There are three implementation levels defined for IMS LD, with level A being the basic and most important one, since it offers the core concepts such as roles, activities, activity structures, and other concepts that are needed to describe a process. Levels B and C add additional concepts to describe more complex models.

Packaging. IMS LD makes use of, or is extensible with other specifications. One of them is IMS Content Packaging (CP) [IMS03, Pul07], which is used to package and describe a learning design model in a way that is transferable between different IMS LD systems. The packages contain an XML manifest file named "imsmanifest". This manifest file can be seen as the entry point to the package, which describes the other resources in the package (text files, binary files, etc.) and also contains an IMS LD representation of the learning design that uses those resources.

Level A Metamodel. To give an impression of the structure of the IMS LD metamodel, the conceptual model of IMS LD level A elements is presented in Fig. 2.14, adapted from [IMS03]. A learning design model contains learning activities (such as 'read a book') or support activities (such as 'supervise learners') to achieve particular learning outcomes using learning environments. An environment is a container that can contain learning objects (a text document, a video, etc.) and services (such as conferencing, mail). The activities are performed by roles, which can either be learner roles (such as "student") or staff roles (such as "tutor"). At runtime, the roles are assumed by particular persons—that is, user accounts in the run-time environment. Activity structures can be used to organize activities as a sequence or a selection (when learners may choose between "read a book" and "watch a video"). The metamodel was built with the metaphor of a theatre play in mind. Therefore,

[6] This section contains content published in [NDK13]

the assignment of roles to activities is made with role-parts (for example the role "student" performs activity "read a book"). Several role-parts are subsumed under an act. The begin and end of an act are used as synchronization points for the role-parts. A play can contain several acts, which are executed in a strict sequence. The learning design model is captured in a method, which contains at least one play.

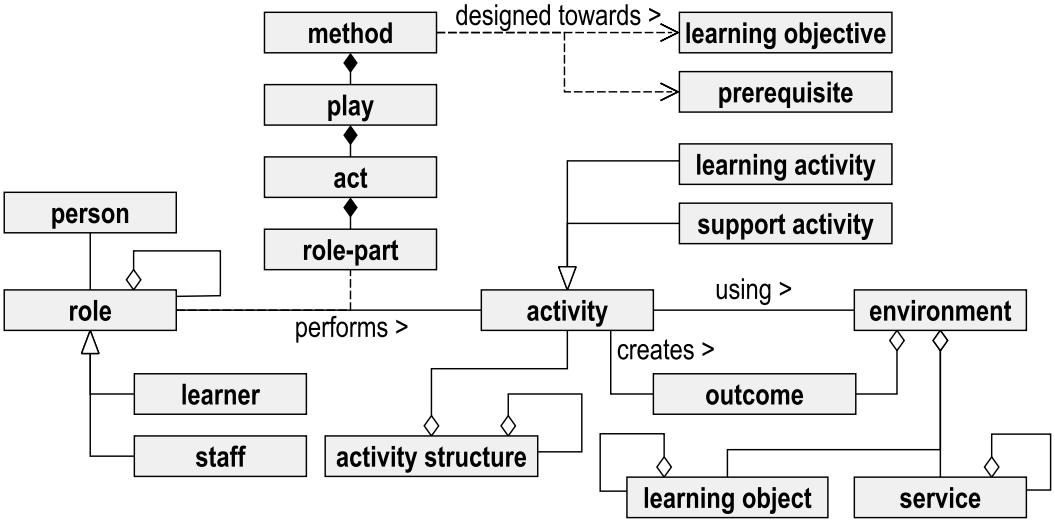


Figure 2.14: IMS LD level A metamodel

Authoring tools. There are several available visual IMS LD tools that simplify the authoring task by providing an easy to use visual modeling environment that does not require the user to write XML documents. The visual modeling metaphors of these tools typically strive to conceal the strict hierarchy and the Web of element references of the XML binding in the graphical user interface. The tools allow to export the native design models as IMS LD packages that could be deployed in any IMS LD compliant run-time environment such as CopperCore.

The available authoring tools have different user interface metaphors. The Reload (Reusable E-Learning Object Authoring and Delivery) tool [Reu04] is a tree based authoring tool, essentially putting a graphical interface on top of the XML tree, and is regarded as a reference implementation of a learning design authoring tool. The ReCourse tool [TEN10] contains a slightly modified structure of the learning design with more intuitive naming and use. OpenGLM [DNO11] is a more recent implementation providing a diagram-based visual modeling environment in an attempt to hide the complexity of the underlying XML representation from the user by providing a modeling metaphor similar to that of UML activity diagrams. OpenGLM also allows the user to import from and export to a remote learning design repository.

In contrast to the previously mentioned tools, which are all desktop applications, WebCol-

lage (Web COLlaborative LeArning desiGn Editor) [HLVA+06] is a Web-based graphical learning design authoring tool based on collaborative learning flow and assessment patterns. It allows practitioners to produce IMS LD compliant learning designs, instantiate them and eventually deploy them in a virtual learning environment.

2.6 Summary

In this chapter, we introduced and defined the most important terms, approaches and technologies used in this dissertation and provided the overall context for our research. Building on CoP and CIS theories, both from works conducted at our department and in the international literature, we presented the methodological framework of our approach and the core domains and scenarios where our investigations took place and where the results contribute to.

In line with our main contributions, we have presented related methodologies, methods and technical realizations. These represent the knowledge to which we relate and compare to when discussing the gaps filled by our studies. With the goal of achieving CoP support for CIS development beyond single communities, we identified several gaps, both of methodological and technical nature. We observed the lack of NRT collaboration adoption across several practices and the still low integration of community members without technical knowledge in design and development stages. The existing Web engineering research showed a focus on classic software development approaches and low integration of Web-based shared editing techniques, due to various reasons such as the rapid change and emergence of new protocols, languages and best practices on the Web. Furthermore, we observed a lack of technical support for rapid prototyping of collaborative, NRT Web-based systems and a need for NRT shared editing libraries that ease the efforts to embed synchronous features into existing or new applications. Next, we present the solutions to these identified problems.

Coming together is a beginning,
staying together is progress, and
working together is success

Henry Ford (1863 -1947)

Chapter 3

Web-based Community Information Systems Methodology

Summary This chapter focuses on the need for support in the development of CIS through methodological and technological approaches that enable CoPs (their members with or without technical knowledge) to actively contribute to the system's creation. It presents a methodology for transforming single-user Web applications into collaborative, widget-based Web applications (first iteration) and continues with an agile cyclic approach for generating CIS, based on models created synchronously by developers and community members. Here, the MDWE process and the metamodel are described, together with our model to code NRT synchronization method.
Contribution RQ1. *Keywords:* MDWE; Widgetizing methodology; NRT collaborative Web applications; Web application reengineering; Cyclic Web CIS development; Web developer support; Community Application Editor. The results presented here have been published in [NK15, dND+16, dNKJ17]. This chapter contains partially information and content extracted from these publications.

Following the main trends and research threads exposed in the introduction and previous section, we base our investigation on developers' need for conceptual and practical support to create collaborative Web applications, that take advantage of the NRT, frontend technologies. In order to achieve this, we conducted two iterations to create a cyclic, agile approach for CIS engineering. First, we investigate based on empirical data the developer support needed to reengineer single-user Web applications to multi user, collaborative ones. Based on this support, we create our proposed method that abstracts the reengineering methodology to the development of applications from scratch or the restructuring of existing ones. Thus, next we present the community developers' feedback gathered during the development of a widgetizing methodology for community Web application reengineering. The study researches the use of Web systems, composed of Web widgets as the frontends and RESTful microservices as backend components. We identified several main requirements,

which generalize the main principles of our development approach and of the developed applications, part of the CISs:

- *Defined process and methodological support* for model-based information system engineering in CoPs (CIS-R1)
- *Community information system metamodel* for ensuring abstraction and interoperability (CIS-R2)
- *NRT collaborative modeling* to allow all members of a community to simultaneously work on the same application model and also provide flexibility and multiple perspectives on the engineering process using dedicated views, where community members can act as various stakeholders. This supports the productivity, as modelers can easily present their ideas in NRT and it increases the learning (and thus evolution) effects in a CoP, since all stakeholders can follow and participate during the whole application development process (CIS-R3)
- *Awareness of user actions*, such that a modeler knows what other modelers are currently working on, on what views/part of the model they perform actions, which actions, etc. (CIS-R4)
- *Support of community features* such as user and group management, synchronous and asynchronous communication, NRT collaboration and shared editing, etc. This requirement is especially relevant for the community applications resulting based on the approach (CIS-R5)
- *Continuous and automatic deployment* of complete applications or components, in order to support the rapid prototyping (CIS-R6)
- *Flexibility* with respect to persistence, used architectures or infrastructures (CIS-R7)
- *Open source* support to make the applications and/or components accessible for as many community members and external contributors as possible (CIS-R8)

Next, we present the widgetizing methodology iteration using the complete design science steps. We also draw and summarize the results which conducted to our collaborative, community-centric and model-driven engineering cycle.

3.1 First Iteration: A Widgetizing Methodology

As depicted in Fig. 3.1, this section covers the first iteration from our design science approach, namely gathering requirements and best practices from online professional CoPs with the purpose to specify our reengineering methodology.

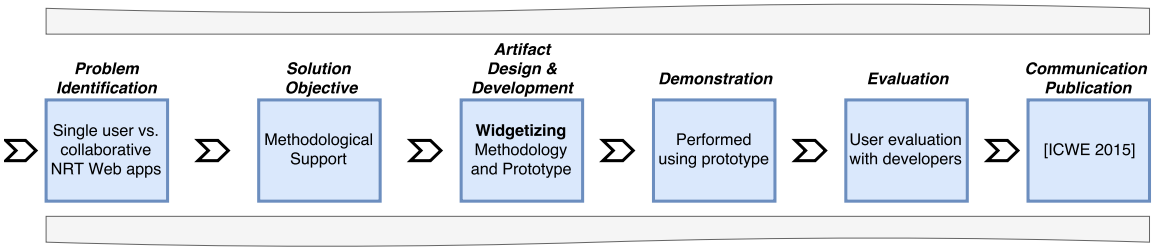


Figure 3.1: Design science approach: A widgetizing methodology

3.1.1 Problem Identification and Solution Objective

Motivated by the rapid evolution of Web technologies for the NRT Web 2.0, there is a reengineering pressure on many existing Web applications for not getting outdated and for reducing costs and maintenance efforts[1]. Existing applications should take advantage of the new enabled opportunities resulting from Web engineering practices such as microservice architectures, continuous integration, statelessness, cloud-based, P2P and avoid becoming obsolete. In this context, developers in online CoPs can benefit from a structured approach to redesign existing applications for collaboration or develop new ones.

Experimental Studies

In the context of our CIS testbeds presented in Chapter 2, we took advantage of existing experience in building widget-based Web applications and reengineering existing community application gathered during engineering semantic video annotation tools for professional CoPs. Therefore, we performed two empirical studies using SeViAnno and SeViAnno 2.0 prototypes, which –besides our own experience from reengineering and implementing the system– offer a good comparison between a single-user and a community, widget-based NRT collaborative Web application.

For this comparison we conducted experimental user studies in a laboratory environment to evaluate the advantages of widget-based applications for real-time collaborative tasks. The goal of these experiments was to identify and measure productivity and collaboration efficiency, via an empirical approach for synchronous collaborative work.

In the first study we used the two versions of the semantic video annotation prototypes. Participants were asked to perform same tasks using the single-user and afterwards the collaborative, widget-based application and to complete a questionnaire after the usage. We used groups of three participants in multiple sessions. The tasks consisted of adding textual annotations, given instructional videos with exact inputs (words and phrases) to be annotated.

[1]This section contains content published in [NK15]

The SeViAnno experiment was planned as a qualitative approach to identify the effectiveness and the collaboration efficiency in two different conditions (versions) when working on the same collaborative task (text-based annotations, cf. Fig. 3.2). The participants attempted

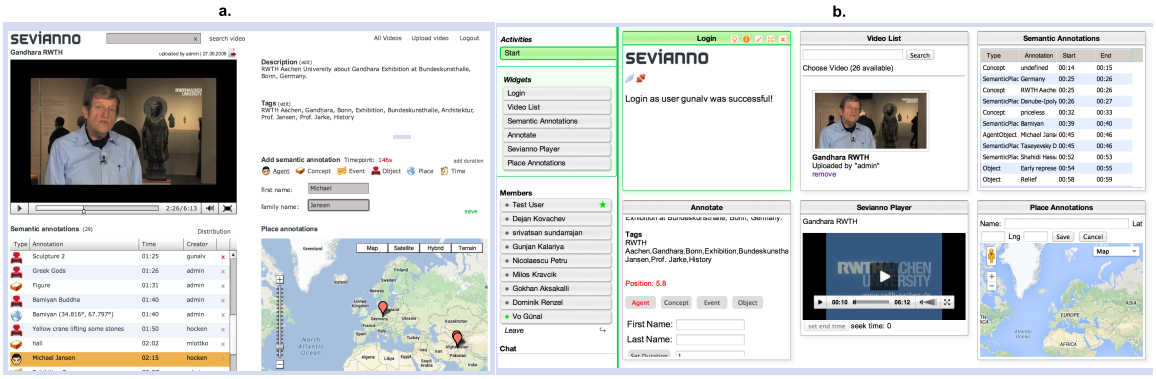


Figure 3.2: SeViAnno (a.) vs. SeViAnno 2.0 (b.)

to add as many as possible correct entries, while also trying to avoid duplicates. We therefore considered the same data entered by different participants to be a sign of inefficient collaboration. We restricted the time to annotate the five-minute long videos to five minutes, and required participants to annotate 60 entries within this time span.

The videos shown in SeViAnno and SeViAnno 2.0 contained different text fragments of similar length. Furthermore, there was a short tutorial with unmeasured demonstrative tasks before the actual task. We had nine frequent computer users participating in this study.

According to the qualitative feedback collected from the results, 67% of the participants thought that instant updates and the environment for work productivity in the widget-based version increased the quality and the value of the work. In addition, 77% of the participants found NRT collaboration with other users via Web widgets to improve the quality of team work. Most importantly, 100% of the participants indicated that the widget-based collaborative version was highly effective to avoid duplications.

A second study was conducted on two different platforms, namely YouTube (using commenting feature for the annotations), and a widget-based prototype designed specifically for this task, extracted from the SeViAnno 2.0 application. In order to further measure the effectiveness and the collaboration efficiency of widgetized applications and their relevance compared to asynchronous social tools, participants were asked to comment on a video, once on the YouTube platform and once using the widget-based prototype. The latter contained a collaborative video player, where the instructional video was loaded and that allowed synchronized broadcasting of the video segment time position among users (same as in SeViAnno 2.0), and a separate NRT widget called “Question and Answer”, that allowed users to add comments in a YouTube like manner. The comments were displayed in NRT among all collaborators.

In each session, three participants added 60 text comments on a given video. We conducted

	Single-user (YouTube)	Widget-based (ROLE SDK)
Total entries	100	58.8
Phrases covered (productivity)	50.6	48.6
Duplication by two users	20.2	9
Duplication by three users	14.6	0.6
Total duplication (excluding originals)	49.4	10.2

Table 3.1: Results of YouTube vs. ROLE-based Application Experiment

five independent sessions with 15 different frequent computer users participants. As in the first study, the goal was to avoid duplicates. The number of correct entries shows the level of productivity, whereas duplicates were considered a sign of inefficient collaboration and low awareness. The phrases used for the two platforms were different from each other to lower the learning effect, and selected from phrase sets for evaluating text entry techniques [MS03]. To reduce eventual dataset noise, we approximated the length of phrases having an average length 22.45 characters for YouTube and 22.71 characters for the widget application. Both applications used the same browser version and Internet connection, during all sessions.

According to the data in Table 3.1, widget applications integrated with NRT collaboration can increase the efficiency and awareness in performing collaborative, synchronous tasks compared to single-user asynchronous social features, such as YouTube comments. In the YouTube scenario, participants lacked the coordination for workload distribution, as the average number of total entries was 100, compared to the widgetized application's average of 58.8. As YouTube does not provide instant updates for seeing others' comments, participants were not able to track their team mates' recent activities in NRT (they did not reload the whole page). In our widget-based prototype participants were able to track others' activities in NRT and thus they could better avoid duplicate work. Quantitatively, participants duplicated 49.4 entries in average on YouTube, while the average of duplicated entries was 10.2 for our application. This shows the strong link between NRT collaboration features and efficiency and awareness.

According to the qualitative feedback collected from the participants, 87% considered that the instant updates in the widget-based prototype increased the quality and value of their comments. In addition, 93% of the participants found NRT collaboration with other users via Web widgets as another very important factor in supporting their team work.

Developer Survey Feedback

Based on literature research of existing Web engineering methods for mashups and Web widgets, we have extracted a set of most important common points and issues suitable for

widgetizing scenarios. Based on these, we performed a user study with widget developers with the goal to sort and detect the requirements for our methodology. The study was conducted using an online questionnaire. In order to engage as many experts in the area as possible, the questionnaire was distributed via mailing lists, forums and email addresses. The targets were communities such as Apache Rave, Shindig, iGoogle as well as researchers that published works about Web widgets in Web engineering conferences. It gathered feedback from 42 members of the widget developer community, ranging from middle to proficient expertise levels, from both academic and industry environments.

Apart from user demographics questions, the questions were divided into

- Considerations, concepts and approaches for designing Web widgets and widget applications (DR)
- Pros and cons for widget-based Web applications (WR)
- Implementation, architecture and limitations of widget-based Web applications (IR)
- User/developer expectations from a widgetizing methodology (UR)

The questionnaire contained both structured and unstructured questions. The structured questions used a one to five likert scale, five representing the best and one the worst rating.

The majority of the participants have been involved in developing Web widgets from scratch (34), Web widget modification (36) and Web applications redesign to widget-based versions (28). More specific tasks include user interface design (38), testing (30), service development (29) and designing the persistence layer (17). The platforms where the participants worked on include Netvibes, iGoogle, Yahoo Pipes, Apache Rave, IBM Mashups and ROLE SDK.

Some of the most important results for the widgetizing design considerations are presented in Fig. 3.3.

The results show that the compatibility of the messages exchanged between widgets (in terms of inputs, outputs, storage, services, etc.) has a high priority for developers. Other important aspects of widgetizing are clearly performance, state preservation and enabling the application to migrate different functionalities to heterogeneous devices. In this case, the support of the widget platform for distribution is necessary, whereas identifying and separating the functionality appropriately for each widget is critical.

We also aimed to gather strengths and weaknesses of widget-based environments. Our findings show that developers want to use widgets because of features such as reusability (WR1. 88%), allowing personalization (WR2. 79%), supporting real-time collaborative tasks (WR3. 67%), instant updates and real-time operations (WR4. 62%). According to 88% of the participants, widget-based applications possess better reusability. Hence, it is logical to say that widget applications are more suitable for easy and efficient improvements and reuse.

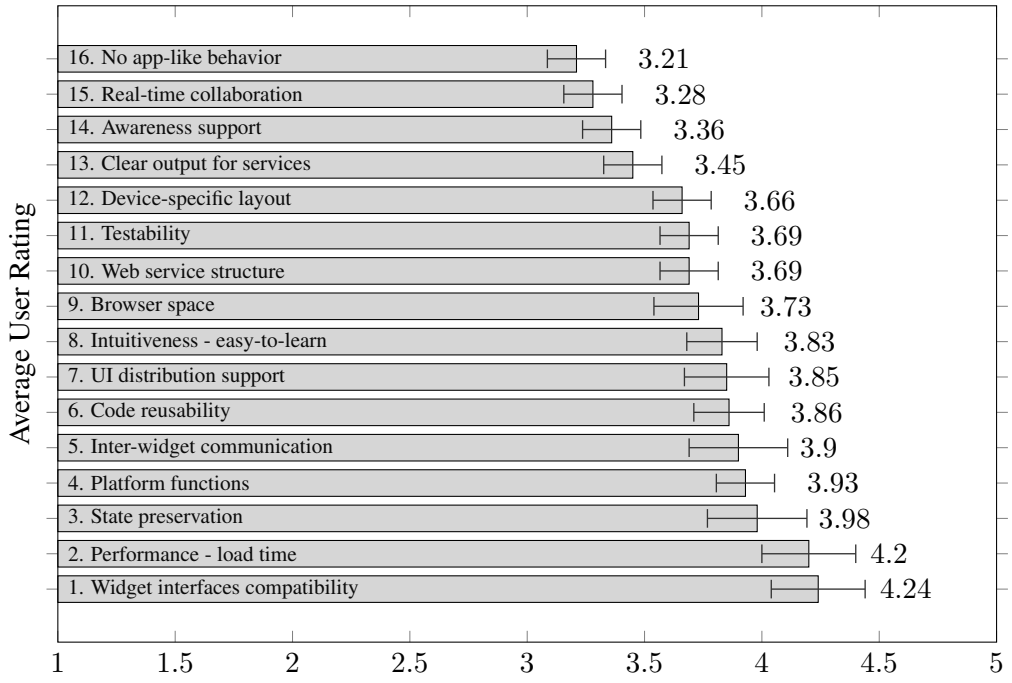


Figure 3.3: Designing widget Web application requirements (DR)

In contrast, 64% of the participants stated that the widget-based applications require more interaction sequences to be tested. However, there are also difficulties to create or reengineer widget applications with the above described features. Regarding the separated implementation of the client-side from the server-side in widget application development, 49% of the participants voted for implementing the server-side first, then design the client-side. 29% preferred a mixed approach and 17% have chosen to design the client-side first, then the server-side. Our results show that developers need support for taking such development decisions. Moreover, 64% of the participants stated that the widget-based applications require more interaction sequences to be tested (WR5.) and that it can be more difficult to use inter-widget communication than the classical Web applications' internal communication (WR6. 71%). These difficulties motivate the need for assistance in implementing the widget-to-widget events, to ease development efforts and ensure consistency of the widgetized application (in terms of interactions and events).

Fig. 3.4 depicts the survey results for the development approach of widget-based Web applications. As it can be observed, following the RESTful architectural style to improve the portability of user interfaces was highly rated.

The last part of our survey, regarding developers' expectations from a widgetizing methodology can be visualized in Fig. 3.5. The highest ratings show the need for documentation and guidance in widgetizing applications (scenarios, guideline). Furthermore, interoperability and standardization is also among the top rated expectations (communication architecture), together with visualization of well-defined steps for reengineering and focus on usability.

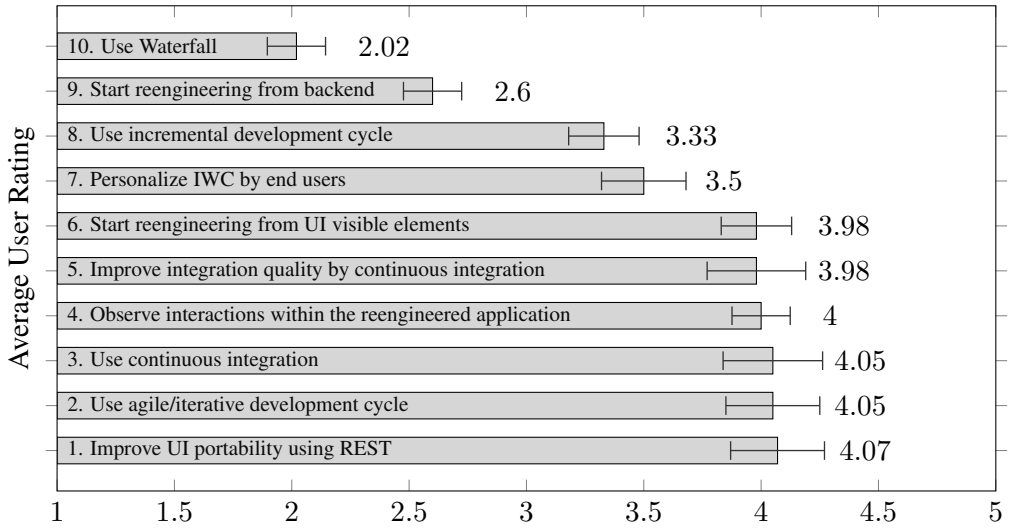


Figure 3.4: Results for implementation approaches requirements (IR)

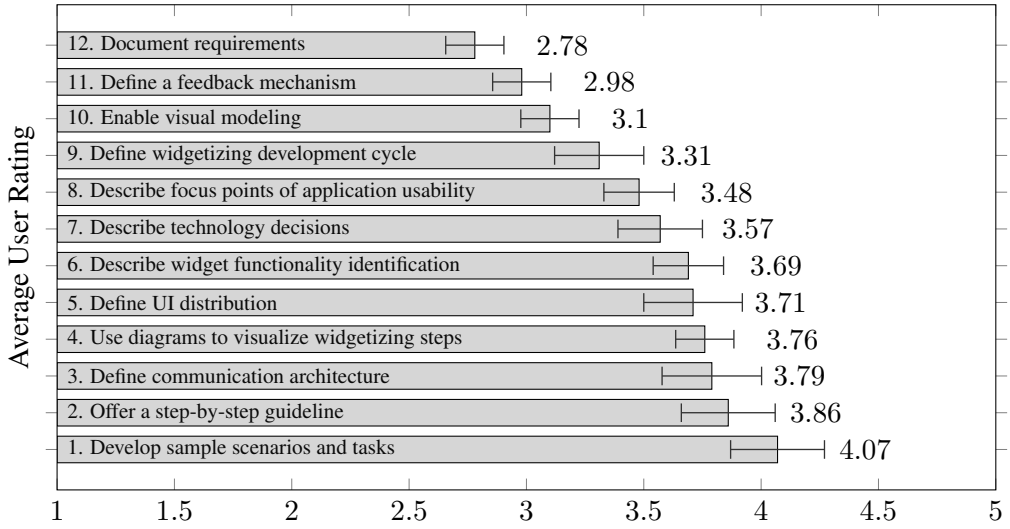


Figure 3.5: Results for user/developer expectations (UR)

The user study qualitative results are an important facet for the requirements of a novel and comprehensive methodology, applicable in widgetizing scenarios. These scenarios cover the development of new widget-based Web applications, as well as the reengineering of existing (single user) Web applications. In order to provide the necessary developer support, we define the methodology and offer design principles and best practices for widget and the design, development and software engineering of Web-based mashup applications. In order to prove the feasibility of a collaborative modeling and MDWE framework based on the methodology for such widgetizing scenarios, we use in this first iteration for demonstration of our methodology principles a prototype based on the resulting widgetizing guidelines.

3.1.2 Artifact Design and Development: Methodology

In the following, we present the major concepts of our proposed methodology and explain the rationale for considering them. We have defined a widgetizing cycle that adopts a simple iterative approach inspired by adaptive software development processes used in the interactive systems literature [Bal08]. The cycle is composed of the steps *Design*, *Implement*, and *Analyze*, as seen in Fig. 3.6. In the same time, the cycle orients itself to the DevOpsUse methodology for CIS, summarized in the previous chapter.

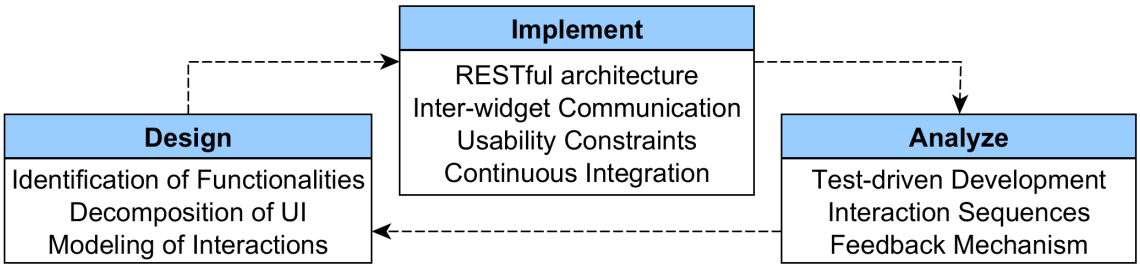


Figure 3.6: Widgetizing cycle

The design step of the widgetizing cycle refers to the conceptual planning and modeling phase for building a widget Web application. The implement step of the methodology specifies an architecture instance based on the concepts explained in the design step. In order to support developer concerns expressed during the initial studies, this step offers solutions for inter-widget communication and collaboration, to be considered during the process of widgetizing. Finally, the analyze step specifies some major concerns for testing and validating the resulted applications. Next, we describe the presented steps in detail, with a larger focus on the design step, as it stands behind the widgetizing editor.

Design Step

During this step, the main functionalities that will be included in the widgetized application should be identified and the user interface decomposed into core, standalone widget components. This decomposition has to minimize the dependencies between the identified functionality, in order to ensure the modularity of the resulting widgetized application and address the coupling between widgets and the identified functionality. Having the widgets and their functionality, the data interactions between them at both interface and microservices levels are to be modeled. In such a case, we consider the functions (their content being defined via inputs and outputs), events and their corresponding interface elements as being part of the lowest needed decomposition level.

As hinted in the overview given above, in a reengineering scenario (in the light of the initial study feedback), we consider the *identification of functionalities* and *decomposition of user interface* to be the major first steps to widgetize an application. For classic Web applications

this can be achieved by analyzing the data relationships and interactions from, to and within both the backend and frontend.

In order to structure the widgets and functions identification and develop the proper conceptual support for the methodology, we have constructed a model (cf. Fig. 3.7) that captures the different elements needed to be specified and related during the widgetizing process.

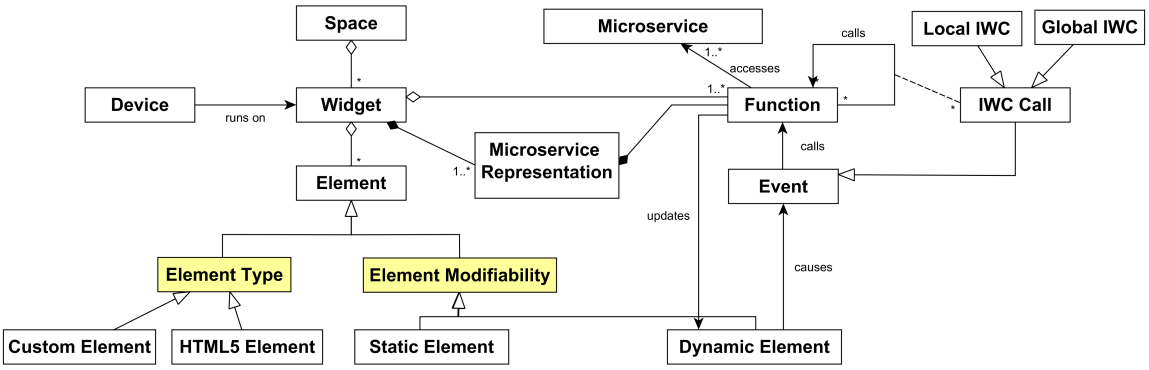


Figure 3.7: A conceptual model of the widgetized platform

A space is the highest dimension of a widgetized platform that aggregates all the widgets selected for widgetizing a given application. The space encapsulates knowledge about the dependencies of its contained widgets and is also responsible for providing a platform for widget-related services. Among such services one can consider the widget rendering container (DR9), inter-widget communication and NRT collaboration and state preservation services, as well as the logic for managing users and devices.

The IWC messages propagation can be achieved across widgets, devices and users and should be available at the space (widget container) level. The state preservation refers to maintaining the state of the various widgets composing a Web application, such that they can be reloaded, distributed across devices, etc. without losing their internal runtime data. As already mentioned in the introduction, the mapping recommendation is that widgets can be used as visualization components for one or more microservices but the number of associations should be kept to a minimum to ensure the resulting widgets follow their definition.

Candidate widgets are most commonly visible via detecting separated panels, headings, navigation bars, embedded contents, and grids. Technically however, for reengineering an application starting from the classic Web application's interface, it is not a trivial task to compute a fully automatic conversion mechanism to identify and decompose the main functionalities of an application using the frontend code.

The microservice representation is a user interface Web component with a core functionality, corresponding to a backend microservice. In order to ensure scalability, rapid prototyping and a modular state-and-device independent architecture, the microservices are assumed to follow the REST principles (such as simplicity, statelessness and uniform interfaces).

A function serves more purposes. For example, it realizes the connection to microservices (within the scope of a widget) for a given microservice representation. Also, functions fulfill utility operations (for example validating the user input on the client side). In this case, they are not attached to a microservice representation but run directly within the scope of a widget. A function performs actions on certain data, which in our context is interesting from the input and outputs perspectives. The inputs and outputs are also related to values of elements. Moreover, functions can also call other functions in the same widget or exchange data at the interface level with other widgets. The inputs and outputs of functions can be processed on the client side only and/or via requests to microservices (such as get, post, put, delete).

The data used inside functions is also important for revealing IWC contents. Once a function is assigned to an element and is connected to a microservice representation and a parent widget, data to be sent via IWC can be identified (that is needed as input by another function linked to a different microservice representation). As an example, in our widgetizing editor, developers were given the possibility to create and assign certain data used by functions and indicate its collaborative or local usage (DR1, DR5).

In the widgetizing context, by elements we refer to user interface components used for rendering (HTML5 components). Our approach distinguishes using view inheritance between different elements, considering their type and modifiability. Based on W3C HTML5 [Hic11] definitions of elements and experience gained from several reengineering projects (SyncMeta and applications for semantic video annotation), we sorted out HTML5 elements that would often give developers guidelines for detecting main functionalities to be identified in decomposed widgets. Basically, these element types can also be mapped to entities from other known Web modeling metamodels from the literature, such as the UIElement from the UWE metamodel [KPZM09], "DisplayUnit" from WebML's hypertext package [SWK06], etc.

Dynamic elements are any user interface elements with an event attribute and/or event listener that evokes an interaction to occur in a Web application. They can process information and/or invoke an activity at the user interface level. As an example, buttons are most common subjects to be identified as dynamic elements (triggers) in Web applications. In a widgetizing context, dynamic elements can also be information-related elements that do not fire an event in the Web application, but only receive data from certain functions. Elements may function with three different purposes, as they either handle inputs via the user interface (text fields, calenders, checkboxes, etc.), or they can be used to display backend info/results (table columns, list items, paragraphs) without evoking an interaction.

Events can be caused from dynamic elements upon fulfilling certain conditions (such as onclick, onchange, onsubmit) that fire an action in the browser. Therefore, a new event can only be assigned to an identified dynamic element in the widgetizing process. Furthermore, once an event is triggered by an element, a function can be called. For designing a new event, developers need to specify its parent trigger, event-handler attribute, and the event handler attribute's value. Thus, we can obtain the *modeling of the interactions* by specifying

such structure of events and functions with their exchanged data.

By performing this modeling, developers will have a clear view of how the events (which can also be identified from the existing traditional single-user application) should be shaped and appropriately distributed in a widgetized application's architecture. In order to let developers manually construct a widget architecture for redesign, we implemented a widgetizing editor following the identification and event modeling principles of the design step.

Furthermore, to support developers in using the editor for performing the identifications, we have also developed a workflow diagram (cf. Fig. 3.9) using PetriNets. This reflects the applicability of the design step and can point to what to follow next (UR1, UR2) based on the conditions and the progress of design. The workflow can be used as a guideline to be followed for modeling each interaction accordingly. In case of new requirements, needed improvements, or new identifications, the workflow can be iteratively applied over existing data, as well as in scenarios of developing a widget application from scratch.

Implement Step

RESTful architectural model for widget development ensures the portability of the microservices and user interface across multiple platforms and scalability of the various components [Fie00]. In the microservice architecture, each service manages its own database. This type of approach for persistence layer suits better to widget-based applications as it can tackle larger volumes of traffic and can scale easily. Furthermore, there is no need for transactions as sharing data of independent Web services can be dealt with by a structured IWC mechanism.

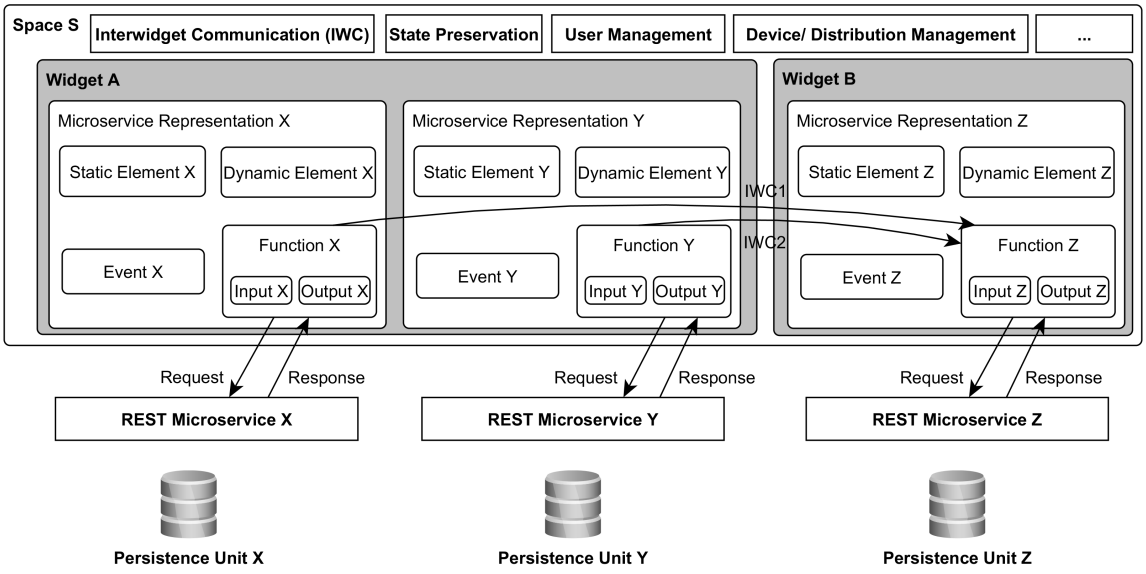


Figure 3.8: An instance of a widgetized architecture template

We integrated the approach of componentization and the conceptual model of our widgetizing methodology to obtain an architecture template (cf Fig. 3.8). According to our proposed widgetizing framework, microservices for each widget run independently from each other. As one single widget can invoke more than one microservice, the services that are used by the same widget work also discrete from each other in terms of persistence.

In a collaborative widget environment, specification of *Inter-widget Communication* (IWC) (DR5) messages and NRT collaboration data types are critical for widget orchestration (DR1), work productivity, NRT updates (DR14) and collaboration awareness (DR15). Collaboration between users is obtained via publishing global IWC messages, which allow browser-to-browser communication.

Modeled interactions in the design step of the widgetizing methodology helps developers to specify the type of IWC messages for each event. Hence, contents assigned as collaborative/ local for an event's function should be implemented and divided accordingly. Developers should consider the aspect of IWC messages in terms of functionality and collaboration awareness. Generally, for each IWC message to published, a description of IWC message based on functionality, IWC type (global or local), data to pass to other widgets and a widget and user identification can be specified [KRN+14]. In general, simple conversions can ensure the data format transformation between various widgets, given that this has been agreed upon.

Separation of IWC messages is also a usability concern for specifying which information to pass to which widget, and what to make visible afterwards in the widget space for a better user experience. In terms of *usability constraints* (DR2, DR12), widgetizing is also an approach related to layout matters. The widget containers often allow users to resize the interface of Web widgets to provide flexibility for a personalized layout. Therefore, the user interface elements that Web widgets contain need to be designed with minimal tooling, in a responsive way. Depending on the tooling in a Web widget, the container is also subject to constraints in the size and layout.

Analyze Step

The analyze step refers to testing and validating widgetized applications. Here, automated tests should be performed, the IWC message interoperability should be validated, the resulted application should be checked against the specified widgetizing model and architecture, etc. Furthermore, improvements and new features should be carefully analyzed and included in the widgetizing cycle. In case of the detection of inappropriate modeling of interactions, wrong widget/microservice representations identifications or other bugs in the specification, developers should follow the cycle by returning to the previous proposed steps.

Test-driven development deals with achieving an environment for automated testing and enabling the iterations of the widgetizing cycle for solving problems and integrating new features. Testing a collaborative widget-based Web application is a challenging task due

to debugging complexity. Therefore, applicability of test-driven development is a well-suited approach for widgetizing scenarios, lowering the risk of errors and bugs while also improving the code quality.

Each *interaction sequence* should be tested after it has been implemented. In order to ensure compatibility of widgets, developers should build test units for all interactions corresponding to the sequence of widget-to-widget communication. Because our process is iterative, regression testing and continuous integration (IR3) should be used to ensure new bugs are not inserted with new iterations and to facilitate rapid prototyping.

How end users perceive a system to improve their practice is critical to its success and therefore opinions from all users (experts/novices, supervisors/workers) are equally valid. Feedback may reveal faults, new identification and decomposition tasks, interaction models and new features to be implemented. Based on previous experience, we recommend the usage of feedback platforms for agile processes and social requirements engineering tools, that allow developers and end users to interact, communicate and work collectively on development issues.

Widgetizing Developer Guidelines in CIS

We developed a workflow diagram seen in Fig. 3.9 via Petri nets to reflect the appliance of the design step for guiding developers what to follow next based on the conditions and the progress of design. Each instance in the diagram is uniquely numbered as we elaborate them below in detail, yet the values in numbering do not represent the sequence of the progress.

1 Design process starts with the identification of functionalities based on the widget hierarchy

2 A component (such as widget, microservice, trigger or variable element) is assigned for specific identification

3 The transition is evoked, if the component to identify is a widget

4 A widget is defined and/or modified with a unique name

5 The transition for assigning children components (microservices and/or elements) for the defined widget

6 The transition is evoked, if the component to identify is a microservice

7 A microservice is identified with a type (such as Form, Table/Grid, or Embed)

8 The transition is evoked, if the identified microservice is already assigned to a parent widget, in order to assign children components (trigger or variable elements) to it

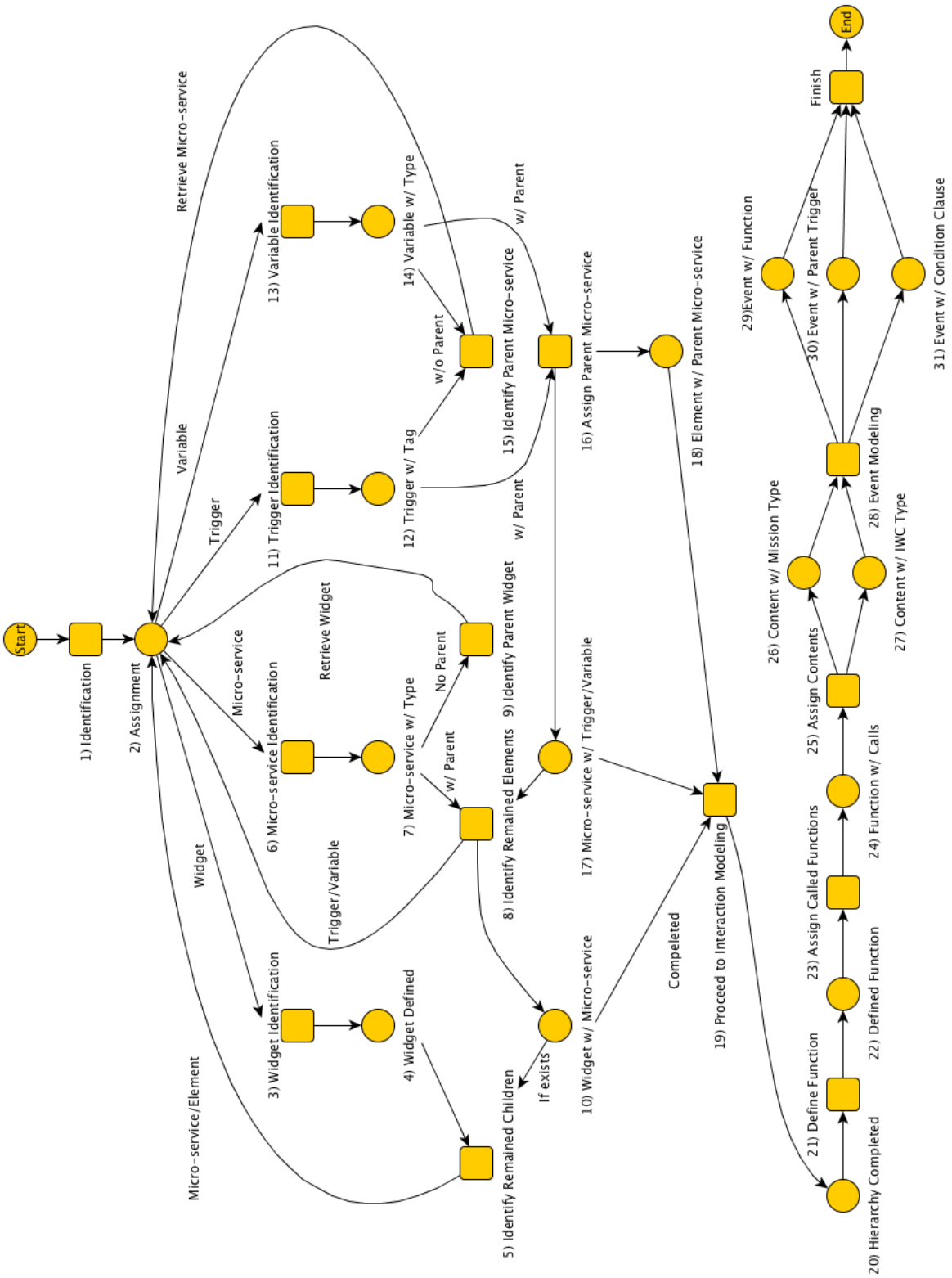


Figure 3.9: Widgetizing workflow

9 The transition is evoked, if the identified microservice is not assigned to a parent widget, in order to assign it as an aggregate part of a new or existing widget

10 The condition where a microservice is assigned to a widget

11 The transition is evoked, if the component to identify is a trigger element

12 A trigger element is identified with an element tag (for example <button >)

13 The transition is evoked, if the component to identify is a variable element

14 A variable element is identified with an element tag (for example <input type=text->)

15 The transition is evoked, if the identified element is not assigned to a parent microservice, in order to assign it as an aggregate part of a new microservice

16 The transition is evoked, if the identified element has a parent microservice to assign it to an existing (already identified) microservice

17 The condition to check whether the parent microservice of an identified trigger and/or variable element needs more children elements to be identified

18 The condition where the parent microservice of an identified trigger and/or variable element has a parent widget and all of its children elements are identified

19 The transition is evoked once the decomposition of all related the widget parts to be used in an interaction is completed

20 The condition where it is possible to model an interaction by using the identified components

21 The transition is evoked to define a function to be called by an event

22 The state where a function with a unique function name is defined

23 The transition is evoked to assign existing functions to be called (such as callback functions) by a defined function

24 The condition where a new function is defined with the information of what to call

25 The transition is evoked to assign contents to the defined function

26 Defined function is assigned with variable or trigger elements as contents with the information of their data related mission, namely HTTP request, such as POST, GET, PUT, or DELETE

27 Defined function is assigned variable or trigger elements as contents with the information of IWC type, such as collaborative or local, to specify the desired awareness that will be resulted by performed the task

28 The transition is evoked to model an event that is triggered by an identified trigger element at a certain condition, and to call a defined function

29 The state where an event has a function call

30 The state where an event has a parent trigger element that evokes it

31 The state where an event has an event attribute (such as onclick, onsubmit) as its condition clause

The workflow can be used as a guideline to followed for each interaction modeling accordingly. In case of new requirements, needed improvements, or new identifications, the workflow can be iteratively applied over an existing model, as well as in scenarios of developing a widget application from the scratch.

3.1.3 Demonstration: A Widgetizing Editor

Based on our methodology we have implemented a prototype in order to evaluate the feasibility of the design step, the completeness of the methodology and to assess to which extent the identifications and the reengineering approach in the design phase can be (semi) automatized. The editor consists of 8 different widgets with specific functionalities, namely *Widgets, Micro-services, Elements, Events, Functions, Contents, Interactions*, and *Architecture* and is presented in Fig. 3.10. Each widget models the concepts explained above. The aim was to provide developers a collaborative and widget-based platform that enables them to try and visualize the design step of the methodology by modeling a well-designed widget architecture. The prototype editor supports developers to model interactions and perform the decomposition of classic Web applications collaboratively, with visualize the widgetized Web application architecture.

The prototype is built using the ROLE SDK. The widgets are developed using HTML5 and JavaScript and Bootstrap to provide responsiveness and easy to use user interface.

3.1.4 Widgetizing Evaluation

For the evaluation of the main concepts of widgetizing methodology and its tool support, we conducted a questionnaire-based user study with the participation of 19 developers and researchers and a face-to-face structured evaluation with 9 widget developers. The questionnaire participants already possessed widget development experience (74%), the rest being end users of such applications. Among the participants were also users that were included in the initial survey. Because we wanted to gather feedback from as many widget experts as possible, the widgetizing methodology and its main steps were described in a video, which also showcased the modeling prototype based on a real-world example. The

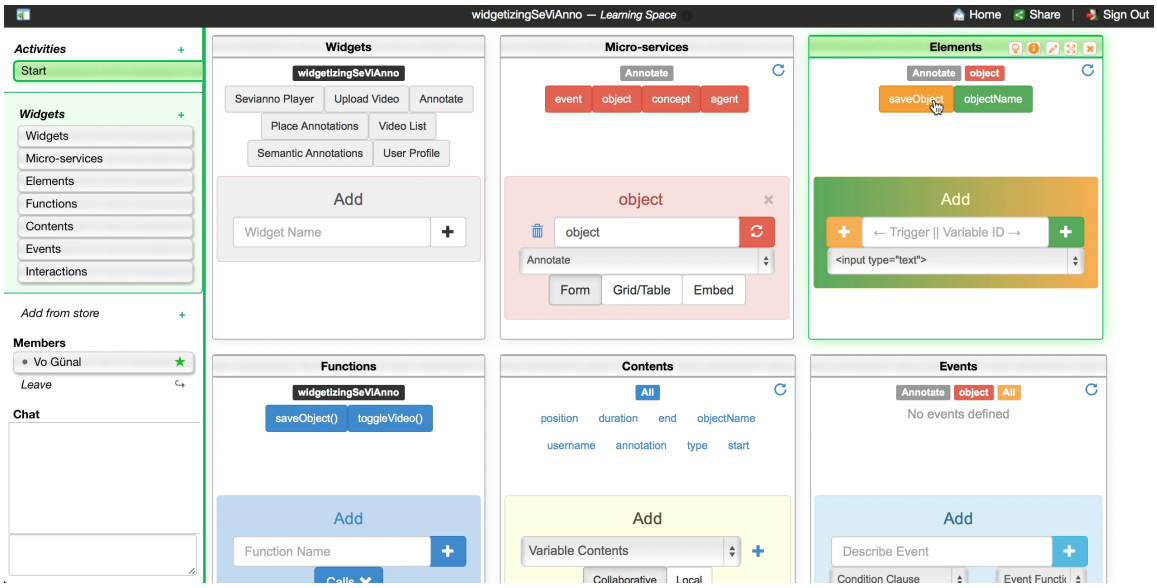


Figure 3.10: User interface of the widgetizing editor

goals were to help the participants to understand the methodology and its realization and assess the fulfillment of the main requirements. The video included the widgetizing cycle steps and an example of a widgetizing scenario – reengineer a semantic video annotation application with the use of the widgetizing editor prototype. The evaluation was performed using a questionnaire with structured questions (based on a one to five likert scale) and unstructured questions. The summary of the results are reflected in Fig. 3.11.

The experimental laboratory study included computer scientists well familiar with widget development. This second evaluation was performed in order to eliminate any confusion or misunderstandings that could have occurred in the first evaluation step and to be able to observe the participants and their actions during the evaluation. Another goal of this evaluation was to study if such a shared editor can help developers in modeling their reengineered applications and if the editor can be a starting point for automatizing the process. For this step, two widgetizing tasks were prepared for the design step to be performed by the participants - both without a time limit. We conducted individual sessions, where each participant was given a 5-minute tutorial about widgetizing, the concepts of our widgetizing model and the editor. After the short tutorial, we provided structured tasks of a widgetizing scenario to the participants. After performing them using the editor, all participants filled out a questionnaire meant to evaluate the methodology in practice. The questionnaire contained structured fields and additional unstructured fields for user comments. For the structured questions, the questionnaire used a one to five likert scale, five representing the best and one the worst rating. The results are presented in Fig. 3.12.

During the online evaluation, the widgetizing methodology was rated with 3.47 from 5. The results show that the main requirements identified in the beginning of our study were

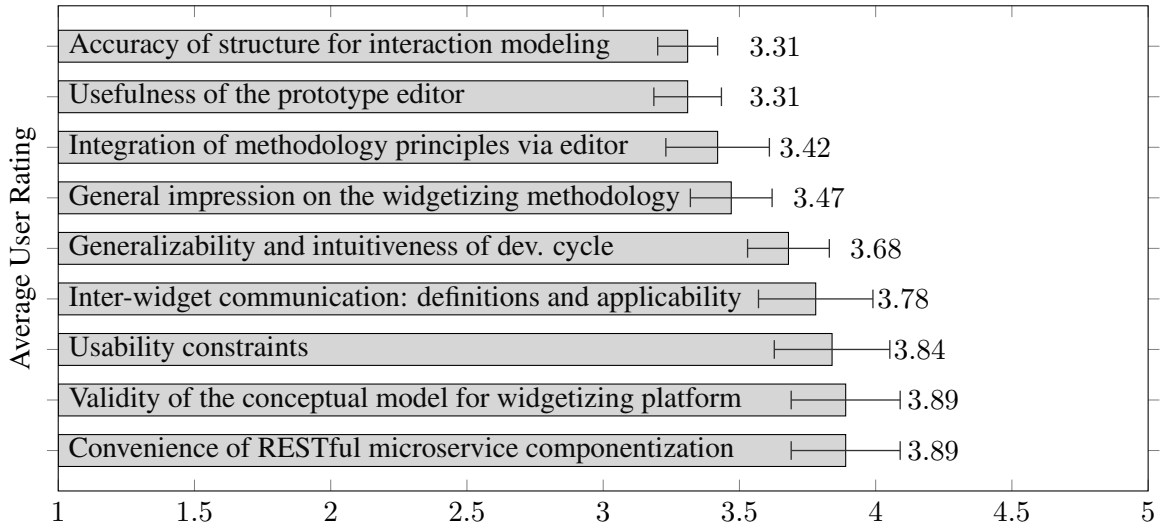


Figure 3.11: Results from the online methodology evaluation

considered satisfactory fulfilled by the participants in the evaluation. Based on the qualitative results collected, the iterative widgetizing development cycle proposed is well suited for modern development scenarios with constant changes of requirements and the needs for improvement, including transforming classic applications to widget-based collaborative Web suites. Despite the limited amount of time and disadvantage of distant presentation, we also got feedback from participants on the widgetizing prototype based on the sample scenario demonstrated in the video. The majority of participants stated that it would be useful to model an intuitive widget architecture in the design step via the presented widgetizing editor that was rated with 3.31 in average.

Moreover, based on the feedback gathered during the hands-on study, developers considered that modeling the widgetized applications in an editor together with an automatic approach for generating the basic structure of such an application would increase the implementation efficiency, lead to better modular, structured applications.

RESTful microservice componentization was considered to be very appropriate by the developers in the context of modular coding and increasing scalability for widgetizing scenarios. Hence, the recommended approaches for implementation of the widgetizing process specified by the methodology can be considered to function as a valid guidance during the designing, coding and testing phases of widget development/reengineering.

Based on the use of the editor on the given task, participants rated the intuitiveness with 3.44 in average. Based on their estimation, the required proficiency level for using the widgetizing editor was above average (3.44), but we consider that the difficulty can be lowered by creating precise and structured user instructions for the editor and by introducing (semi) automatic steps for connecting the elements, functions and events in the editor. Via comments some of the participants requested the increase of intuitiveness and user

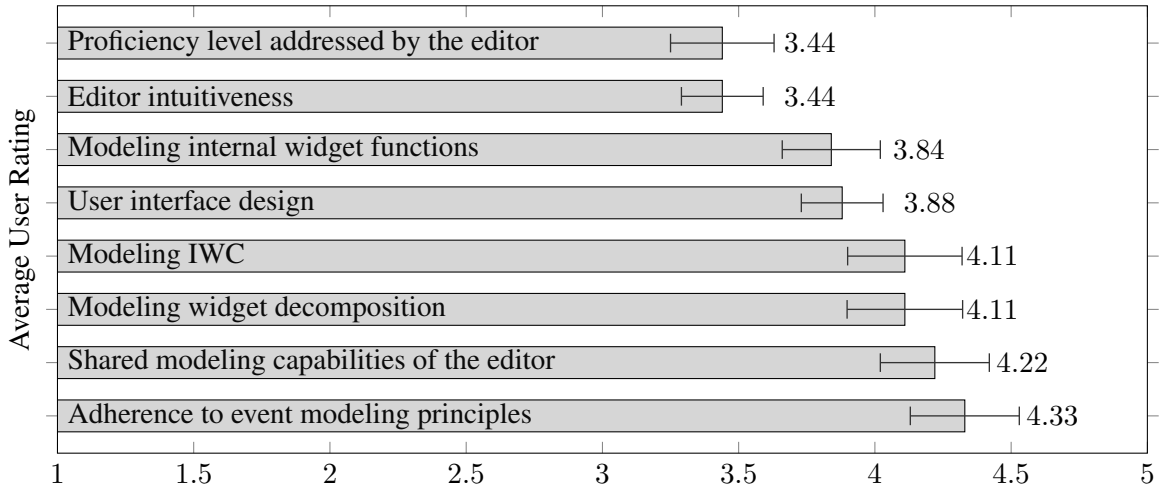


Figure 3.12: Qualitative user feedback on widgetizing editor

awareness by features that enable users with interactive recommendations, that can support them as they interact with the editor. We consider that this can be achieved during the modeling phase, by logging and presenting the different modeling actions of users and by providing NRT collaboration support with awareness features.

The resulted model of the widgetizing step via the editor was rated 4.11 in average. Thus, 77% of the participants found the model they constructed very convenient, offering a good overview of the decomposed widget application design. Furthermore, the model for designing IWC via the editor was also rated 4.11. Participants found it easy to point to IWC messages to be implemented based on the hierarchical information of interacted data and interacting functions of events. The participants rated the event modeling concepts with 4.32 in average. These results support that the prototype provides developers very helpful information based on the decomposition and event modeling phases of the design step of our widgetizing methodology, enabling them to have a well-designed widget architecture model for the implementation step. Designing the architecture in a shared modeling space via an editor was considered to be a well-suited idea to have well-designed widget models by 88% of the participants, as it was rated with 4.22 in average. As such, the majority of widget developers believed that designing architecture models in a shared platform would increase the accessibility, designer/developer awareness and communication, and the quality of the widgetized model. The results show a positive influence of the widgetizing methodology upon widget application development and a high impact of the widgetizing editor upon performing the widgetizing steps.

Communication: Discussion and Lessons Learned

The methodology was published in [NK15] and presented in front of the academic community at the International Conference on Web Engineering 2015. In Table 3.2, we summarize

the requirements fulfilled through the methodology and based on its findings we discuss the take away messages which were also considered as an entry point for our model-driven Web engineering approach for CIS development:

Table 3.2: Requirements fulfilled using widgetizing methodology

Defined process	*CIS metamodel*	*Conceptual modeling*	*Awareness*	*Support for CoP features*	*Automatic deployment*	*Flexibility*	*Open source*
R1	R2	R3	R4	R5	R6	R7	R8
●	●	(●)	(●)	(●)	x	x	x

The methodology guides developers to redesign classical (single-user) Web applications as widget-based collaborative Web applications. Furthermore, it guides to achieve interoperability and rapid prototyping. The evaluation results show that the methodology and a related widgetizing editor can be successfully used to design and specify such widget applications. Together with the step-by-step guideline, they are offer a structured approach in creating and visualizing widgetized systems and were received positively by the community. Overall, it eases the implementation efforts in widget developer communities, helping them to create relevant models of widgetized applications.

The widgetizing metamodel also allows for a generalization of widget-based Web applications and offers a common ground for their architectures. In general, especially for reengineering tasks, widgetizing scenarios are beneficial, considering best practices and reflection upon design. They pave the way to a new approach in designing and implementing applications for various communities. However, due to its specificity and rather technical nature, the proof-of-concept prototype can only be used by widget developers and has therefore a rather narrow applicability in CoPs. The empirical results also show that it is hard to enforce such practices on developers, that for example might prefer a top-down design approach to a bottom up one. Even more, developers require automation and support through the entire software lifecycle. Agile approaches need a tight integration of end users in requirements gathering and elicitation and during application development. As a solution to these challenges, we concentrate in the next iteration on providing a flexible and NRT application modeling technique, that can be used by both developers and community users without technical knowledge and enable further automation in the engineering process, such as code generation, using existing paradigms and code templates (for example microservices and widget structure, communication interoperability). Since the methodology best practices implemented in the prototype lead to a Web application architecture description, this step was also suggested during the evaluation, namely to use models as artifacts in community design processes and generate applications directly from the results. Code generation of microservices, widgets and complete systems should be possible at any time during the modeling process, provided that the current state of the application's model is valid. Moreover, the collaboration could be also improved, by adding more awareness, NRT

capabilities and the possibility to collaboratively model the applications, by a more flexible approach in terms of metamodel, appearance and community support.

The relation between the requirements listed before, research questions and the opportunities described in Chapter 2, Fig. 2.5 are depicted in Table 3.3. Based on these opportunities and the results from the first iteration, we identify the following points, in line with our initial requirements, which we address via an agile, collaborative MDWE approach, namely the Community Application Editor (CAE).

Table 3.3: Solution objectives after first iteration

Opportunity	*Requirement*	*Description*	*Research question*
Architectural choice	R2, R4, R7	Allow application logic and interactions both on frontend and backend tiers, using P2P and client-server technologies	RQ2
Rapid prototyping	R6, R7, R8	Support developers, designers and community users to create widgets, services and applications based on Web application models. Add live coding and deployment features on the Web browser and reduce time to deployment once a valid model has been authored	RQ1
Abstraction / Modeling	R2, R3	Generalize the reengineering to Web information systems engineering and enable specific stakeholders to work on specific views of the model, according to their technical experience	RQ1, RQ3
Methodology	R1, R2	Use the developer studies' results to achieve a more participatory-oriented software design lifecycle	RQ1
Collaborative editing	R5	Based on the P2P and client-server architectural choices, NRT shared editing features should support collaboration in CoPs. Make possible the transformation of existing Web applications into collaborative ones, as easy as possible. Add NRT shared editing to the modeling and the code generation phases	RQ2
User guidance	R4	Add awareness during the NRT collaboration and for the collaborative actions involved in the MDWE cycles	RQ3

Therefore, in the second iteration we implement a widgetizing editor with full support for CoP developers and members, where widgets and microservices can be modeled collaboratively, in order to generate collaborative Web applications that benefit of the principles already exposed in this work. For this purpose, we concentrate on the following points:

(a) A MDWE process that supports agility by combining view-based NRT conceptual modeling together with code generation and deployment facilities

(b) Improvement of developer support and training and a further step towards integrating all stakeholders in the development of community Web applications

(c) Architectural and technology support for such an approach, available as a Web-based editor for online CIS

(d) Technology to enable NRT collaboration of models and code levels, as well as easy integration into CIS

Next, we present the conceptual design of the CAE approach, focusing on the CIS development lifecycle, a general metamodel for CIS and the agility enabled by NRT synchronous collaboration processes and the cyclic approach realized through code and model synchronization.

3.2 Artifact Design and Development: An Agile MDWE Process

This section, as it can be seen in Fig. 3.13 presents the conceptual, design considerations for building our CIS MDWE approach.

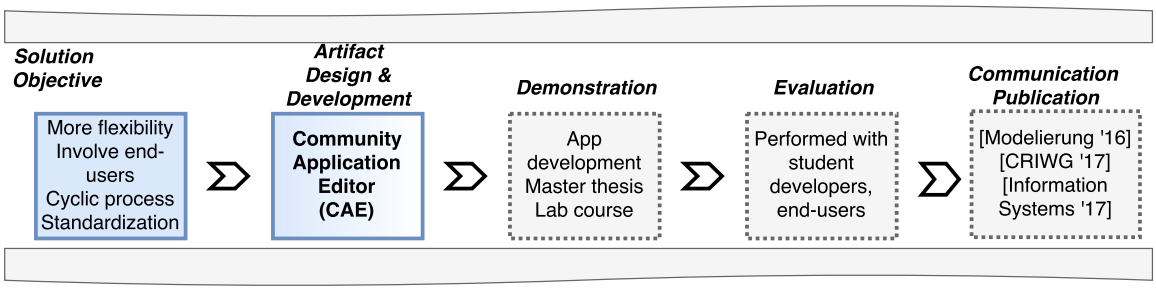


Figure 3.13: Design science approach: Community Application Editor design and development

The Community Application Editor[2] specifies a synchronous, agile development approach for building community, collaborative Web applications. Starting from the widgetizing scenario, we hereby propose an abstraction to develop modular, component-based CISs, based on Web widgets as frontends and RESTful microservices as backend components. The approach contributes with an agile development cycle, a faster way to obtain training in professional online CoPs and deal with the heterogeneous software landscape of CISs (in terms of reducing complexity and increase comprehension, extensibility and reusability).

The generated source code also provides a unified structure, based on templates for both backend and frontend application components. By using the CAE for the generation and extension of community Web applications, the CoP creates a common ground for development and a unified software landscape. Using well defined means of communication

[2]Part of the contents of this section have been published in [dNKJ17]

and collaboration, the generated applications are easy to comprehend for new developers and end users, which builds the basis for easy extensibility.

Fig. 3.14 presents our cyclic approach for rapid prototyping of our targeted widget-based Web applications. The cycle is achieved by synchronizing collaborative modeling phases and live source code editing.

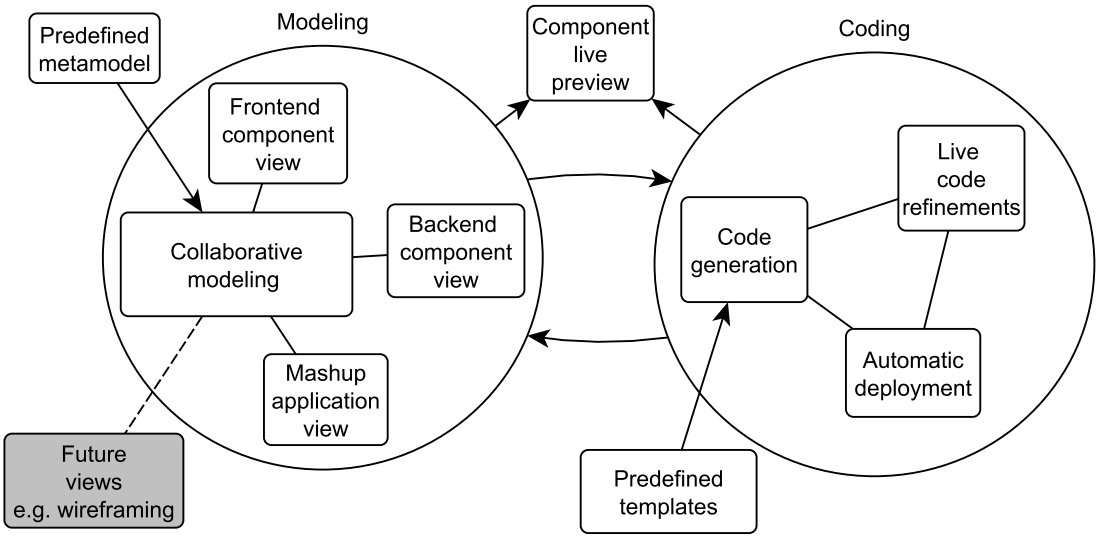


Figure 3.14: Collaborative application modeling and code generation cyclic approach

Both modeling and coding are performed on the Web in a NRT collaborative manner. Therefore, changes in the model and source code are directly visible to all involved collaborators, at all times. It is also possible to work on one of the two phases at any point and for different teams to work simultaneously on the code or model. All changes on the model level are immediately reflected in the generated source code. We ensure that the models are not broken by code changes and enforce a certain base architecture through the usage of protected segments.

Protected segments in the source code describe a functionality that is reflected by a modeling element. In the modeling phase, based on a unified metamodel, community members, assuming various roles, can specify the architecture of the system using multiple views. We define three views of a Web application: the frontend, backend and the application mashup respectively.

The model is transformed into code based on predefined open source templates for widgets and microservices. Source code editing, referred to as code refinements is taken into account upon model-to-code regeneration, integrating the changes accordingly into the regenerated source code. The resulting frontend functionality is displayed synchronously with the changes in a live preview. This functionality makes it much easier to get feedback on the impact of performed changes.

Fig. 3.15 shows the main activities of our Web application's creation process. The authoring of models and code brings together community members grouped into teams and behaving as different stakeholders (developer, architect, designer, etc.).

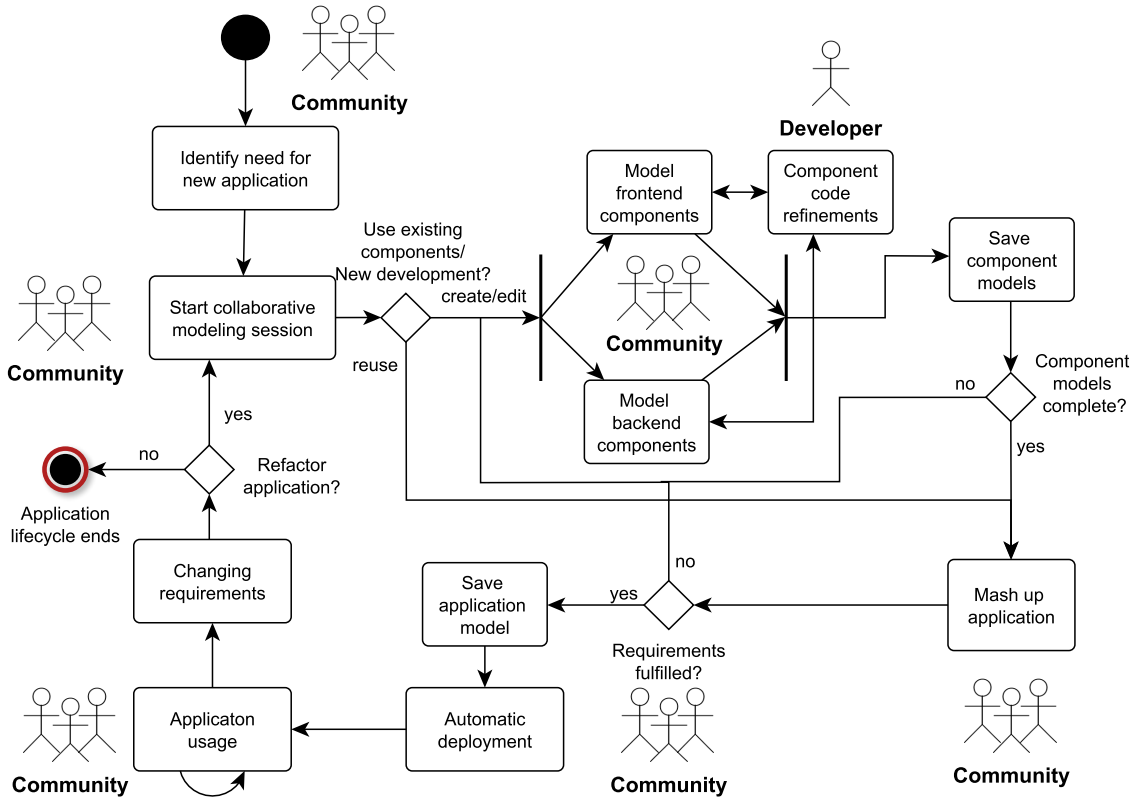


Figure 3.15: Collaborative application modeling and generation process

The first step is the CoP identifying the need for a new application or the need to extend an existing one. There are approaches which cover the early community requirements elicitation processes. An example of such a tool is the Requirements Bazaar [RBKJ13], which enables a social, community-oriented requirements gathering practice.

Since the targeted group for our tool is a whole professional community, users have various roles, such as developer, software architect or non-developer community member. Once the modeling process begins, it is split up into two phases; The first phase is the "collaborative application modeling", where all members of a CoP are able to bring in their ideas by collaboratively modeling the application. The second phase is the "code refinement phase", where members of the CoP experienced in development of Web applications work on the application's source code, until it fully matches the needs of the community. Such fulfillment of requirements can be negotiated in NRT over the Web directly using the editors or using communication means such as a chat, which is provided directly in the modeling space, or other (external) means (such as video chat).

The predefined metamodel is split up into three different views to allow for a componentized architecture that can be developed concurrently, also allowing to reuse components for multiple applications. Modelers thus can work collaboratively or in parallel for defining the client- and server-side components, for example define widgets, microservices, databases and interactions.

The collaborative application modeling phase itself can again be roughly split up into the development of the basic components the application should consist of (widgets and microservices should be created in the dedicated views) and –secondly– defining their dependencies (mashup view). Once the microservices and widgets have been defined and the participants are satisfied with a current state of an authored model, this can be persisted. Persistence also triggers the code generation for the corresponding client- or server-side components. The outcome of this step is a basic code framework reflecting the model.

This code basis then has to be evaluated and compared to the requirements of the CoP and if they are fulfilled, the application mashup modeling phase starts. If not, the application and its model have to undergo an additional modeling/editing phase, until they match the requirements. The "code refinements" phase can again be roughly split up into frontend (widget-related) work and backend (microservice-related) work.

We apply related work on traceability [OO07] and synchronization [HLR08] from the model-driven engineering domain to formalize our agile collaborative MDWE integrity constraints. In order to adapt the synchronization techniques to the NRT collaboration setting, a trace model providing linking information between model elements and source code artifacts was developed. In this phase, complex logic is implemented, which can not be easily depicted in the application model itself. Members can choose at any point to generate the Web application and see the (preliminary) result in the Mashup/Communication view.

Once the community members agree, the mashup (meaning a full-fledged Web application model) can be persisted and the final application is generated. The code is available as open source in a repository of the community. Finally, our approach features the possibility to automatically (re-)deploy an application as soon as its code was (re-)generated to support continuous deployment.

3.3 A Metamodel for Community Applications

Our approach is based on the widgetizing metamodel developed in the first iteration. This is extended with a microservice-based backend metamodel and NRT collaboration elements, in order to reflect a complete Web-based CIS. Fig. 3.16 depicts the metamodel of the CAE concept. It is composed from three different views: the frontend view (*Frontend Component*), the backend view (*Microservice*) and the mashup/communication view, which allows for an easier, more fine granular development of an application. Multiple microservices and frontend components form an application environment, which runs on certain devices and is

used by community members. The community, its members are agents, according to the ATLAS methodology.

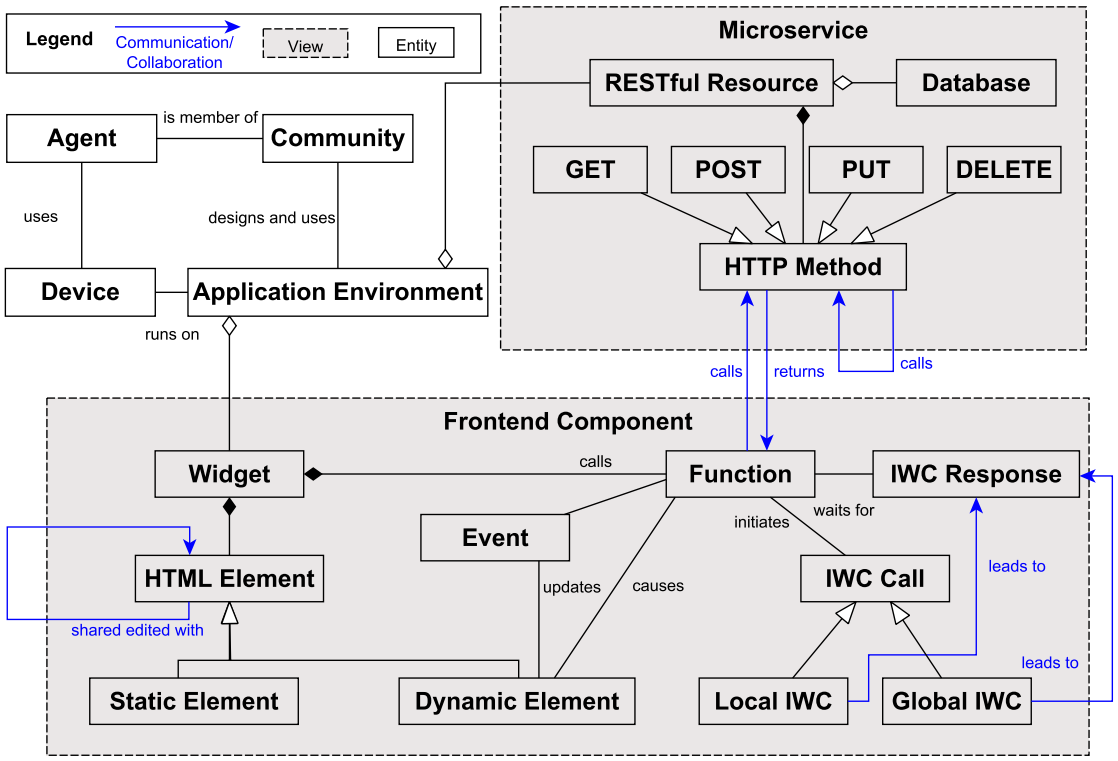


Figure 3.16: Community information system metamodel

The Microservice View. The central entity of the microservice view is a RESTful resource. It can have access to a database and contains HTTP methods, which form the interface for communication either via a RESTful approach, but also via an internal service call from one HTTP method to another (of a possibly different service and/or server) for orchestration purpose. These calls use an underlying P2P network for an easy and fast way of communication. According to the idea of polyglot persistence and microservice architecture principles, each microservice has access to its own database instance.

The Frontend Component View. The central entity of the frontend component view is a Widget. This consists of Functions and HTML Elements. HTML elements can either be static, meaning that they are not modified by any other element or functionality of the application, or dynamic, meaning that it either is created or updated by one of the frontend component's elements. A function is able to update or create a dynamic HTML element. Both static and dynamic HTML elements can trigger events, which could for example be a mouse click, that cause function calls. The second option to trigger a function call is via communication means at the frontend level, by IWC calls. These function similar as

android intents, offer an encapsulation of a certain communication protocol and can transmit data and commands for complex logic between various widgets. IWC can be used for implementing logic and orchestrating widgets without the need of data exchange with the microservices. An IWC Response object waits for an IWC Call to be triggered. These calls are again part of a function, which initiates them. A function is able to update or create a dynamic HTML element. The last part of the frontend component view are the communication and collaboration functionalities, which include the already mentioned IWC call response mechanism, as well as microservice calls, that are triggered by a function, and HTML elements instrumentalized with NRT collaboration/shared editing support. The shared editing support allows for HTML elements to be used collaboratively by multiple users. An example is represented by a video player that has the same state (play or pause) across all devices and community members using it, a checkbox, a text area element that allows for shared text editing, etc.

The Communication View. In a unified community application architecture, the communication means between different components have to be well-defined. The communication view on our metamodel builds the bridge between the frontends and backends. It features a simplified view on the whole application, only depicting those objects directly involved in inter component communication processes. This includes microservice calls from frontend components to microservices, IWC call and response objects as well as collaborative HTML elements. The CAE offers four ways of communication and collaboration through four different mediums:

- Internal microservice method invocation: the internal communication between two microservices is realized by using P2P messages between microservices, that can be sent directly through the network via remote method invocation, without the need to use an indirection via a HTTP (RESTful) call.

- Widget to microservice communication: the frontend to backend is realized via asynchronous RESTful calls. The RESTful requests are initiated by a widget's function.

- Inter widget communication: we use an abstraction over standardized communication protocols in order to send messages between widgets. The two metamodel objects called IWC Call and IWC Response enable modelers to depict the communication between widgets in a formalized way.

- HTML element collaboration: by specifying a HTML Element object as "collaborative", modelers specify that this will be used collaboratively by community members in NRT . This means that in the generated application every change to the element is propagated in a NRT fashion and will have the same state among all members currently watching or modifying it.

3.3.1 Model Synchronization Strategy

As presented before, a part of our cyclic approach is the synchronizing of collaborative modeling phases and live source code editing. For a better understanding of this synchronization, we illustrate the idea on an excerpt of our Frontend view of the CAE metamodel, as shown in Fig. 3.17 . This contains the three elements HTML Element, Event and Function, their attributes and their connections between each other.

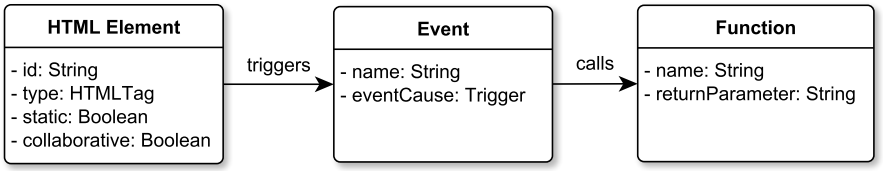


Figure 3.17: Example model

The model synchronization is depicted in Fig. 3.18. It is divided into the synchronization between the source code and its trace model and a second synchronization between the source model and the source code. In the following, we explain our synchronization concept by using a simple formalization. We denote the source models by S_i, source code models by T_i and trace models by tr_i. The source- and source code-metamodels are denoted by M_S and M_T. We use the definition of synchronization expressed in [HLR08], as follows: two models A and B with corresponding metamodels M_A and M_B are synchronized, if

$$trans(A) = strip(B, trans) \tag{3.1}$$

holds for the transformation $trans : M_A \rightarrow M_B$ and a function $strip : M \times (M_A \rightarrow M_B) \rightarrow M$ that reduces a model of M with either $M = M_A$ or $M = M_B$ to only its elements relevant for the transformation. This definition uses the trans and strip functions [HLR08]. Intuitively, the $trans$ function expresses that applying a transformation to the source model yields the target model. The function $strip$ is used to remove any additional elements and map models to only the relevant source/target model. As an example, consider the *height* attribute of an HTML *img* tag. As it can be seen in Fig. 3.17, the HTML Element of our metamodel does not contain a height attribute, so this manually added attribute would not be part of the stripped model according to the defined transformation.

Synchronization of Source Code and Trace Model

Based on a first model S_1, an initial generation of the source code T_1 and its trace model tr_1 is performed. As depicted in Fig. 3.18, the trace model tr_i is updated once the source code changes. ΔT_{2i-1} are applied to the source code T_{2i-1} in the i-th code refinement phase.

Formally, a single source code change can be denoted by one of the two functions $\delta^+_{M_T} : M_T \times C \times \mathbb{N} \rightarrow M_T$ and $\delta^-_{M_T} : M_T \times C \times \mathbb{N} \rightarrow M_T$. While the former inserts a character

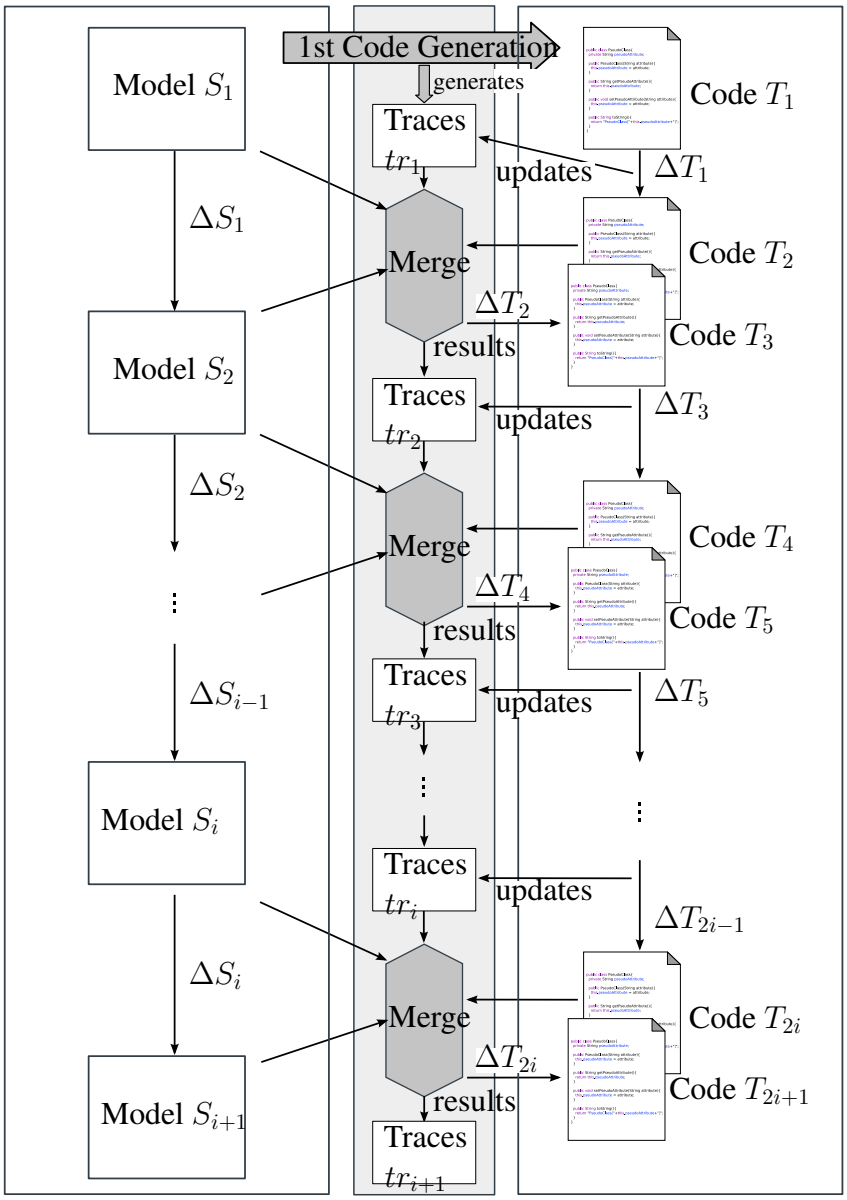


Figure 3.18: Model synchronization

$c \in C$ at position $n \in \mathbb{N}$, the latter deletes a character c from position n in the source code. Then, the result of applying the source code changes ΔT_{2i-1} on T_{2i-1} is defined by:

$$T_{2i-1}\Delta T_{2i-1} := \delta^{\pm}_{M_T}(\delta^{\pm}_{M_T}(\cdots \delta^{\pm}_{M_T}(T_{2i-1}, c_1, n_1)\cdots, c_{k-1}, n_{k-1}), c_k, n_k) =: T_{2i} \quad (3.2)$$

for $n_k, \in \mathbb{N}$, $c_k \in C$ and $k \in \mathbb{N}$.

According to Eq. 3.1, the condition $trans(T_{2i}) = strip(tr_i, trans)$ must hold for the

synchronization between the updated source code T_{2i} and the trace model tr_i:

$$trans(T_{2i}) = strip(tr_i, trans) \tag{3.3}$$

$$\iff trans(T_{2i-1}\Delta T_{2i-1}) = strip(tr_i, trans) \tag{3.4}$$

$$\begin{aligned}\iff & trans(\delta^{\pm}_{M_T}(\delta^{\pm}_{M_T}(\cdots \delta^{\pm}_{M_T}(T_{2i-1}, c_1, n_1)\cdots, c_{k-1}, n_{k-1}), c_k, n_k)) \\ & = strip(tr_i, trans)\end{aligned} \tag{3.5}$$

For the synchronization between source code and trace model, we only need to update the lengths of the segments of the trace model. Therefore, we assume $strip(tr_i, trans) = len(tr_i)$, where $len(tr_i)$ is a tuple containing the segments' lengths. This leads to the following equation that must hold after the source code was updated:

$$trans(\delta^{\pm}_{M_T}(\delta^{\pm}_{M_T}(\cdots \delta^{\pm}_{M_T}(T_{2i-1}, c_1, n_1)\cdots, c_{k-1}, n_{k-1}), c_k, n_k)) = len(tr_i) \tag{3.6}$$

To satisfy this condition, each source code change needs to update the length of the segment that is affected by the deletion or insertion. Therefore, each $\delta^{\pm}_{M_T}$ is transformed to an update of the trace model tr_i as follows:

$$\begin{aligned}&\delta^{\pm}_{M_T}(\delta^{\pm}_{M_T}(\cdots \delta^{\pm}_{M_T}(T_{2i-1}, c_1, n_1)\cdots, c_{k-1}, n_{k-1}), c_k, n_k) \rightarrow \\ &\delta^{\pm}_{len}(\delta^{\pm}_{len}(\cdots \delta^{\pm}_{len}(len(tr_i), n_1)\cdots, n_{k-1}), n_k)\end{aligned} \tag{3.7}$$

with

$$\delta^{+}_{len}((l_1, \cdots, l_m), n) := (l_1, \cdots, l_j + 1, \cdots, len_m) \tag{3.8}$$

$$\delta^{-}_{len}((l_1, \cdots, l_m), n) := (l_1, \cdots, l_j - 1, \cdots, len_m) \tag{3.9}$$

where $l_i \in \mathbb{N}$ for $i, m \in \mathbb{N}, 1 \leq i \leq m$ is the length of the i-th segment and $l_j, j \in \mathbb{N}$ for $1 \leq j \leq m$ is the length of the segment that is affected by an insertion or deletion in the source code at position n.

Synchronization of Model and Source Code

In the model synchronization process, the last synchronized model S_i, the updated model S_{i+1}, the current trace model and the last synchronized source code T_{2i} are involved. By using the trace model of S_i, the applied model changes ΔS_i can be merged into the last synchronized source code T_{2i} without overwriting already implemented code refinements. As a result of the model synchronization, the updated source code T_{2i+1} and its trace model are obtained.

In general, model changes can be defined as functions of the form $\delta : M_S \rightarrow M_S$. More specifically, the model changes can be denoted by the following five functions, adapted

from [HLR08]:

$$\delta_t^+ : \text{creating element of type t} \tag{3.10}$$

$$\delta_t^- : \text{deleting element of type t} \tag{3.11}$$

$$\delta_{e,s1,s2}^+ : \text{adding edge from element s1 to s2} \tag{3.12}$$

$$\delta_{e,s1,s2}^- : \text{deleting edge between elements s1 and s2} \tag{3.13}$$

$$\delta_{a,s1,v}^{attr} : \text{setting attribute a of element s1 to value v} \tag{3.14}$$

As such, applying ΔS_i to S_i can be defined as a sequence of these changes:

$$S_i \Delta S_i := \delta_1 \circ \cdots \circ \delta_n(S_i) =: S_{i+1} \tag{3.15}$$

According to Eq. 3.1, the following equation must hold for the synchronization between model and source code:

$$trans(S_{i+1}) = strip(T_{2i+1}, trans) \tag{3.16}$$

$$\iff trans(S_i \Delta S_i) = strip(T_{2i+1}, trans) \tag{3.17}$$

$$\iff trans(\delta_1 \circ \cdots \circ \delta_n(S_i)) = strip(T_{2i+1}, trans) \tag{3.18}$$

Furthermore, as all parts of the source code that directly correspond to model elements are contained in protected segments, we assume $strip(T_{2i+1}, trans) = prot(T_{2i+1})$, where $prot(T_{2i+1})$ represents the source code that is reduced to the content of its protected segments. Finally, this leads to the following equation that must hold after the synchronization process:

$$trans(\delta_1 \circ \cdots \circ \delta_n(S_i)) = prot(T_{2i+1}) \tag{3.19}$$

To satisfy this equation, each individual model change $\delta_i, i \in \mathbb{N}, 1 \leq i \leq n$ is transformed to corresponding source code changes. Before we define the transformations of model to source code changes, we first introduce formulas that are needed for the later transformations.
Attribute value: the value of the attribute labeled *name* of a model element elm is denoted by $attr_{name}(elm) := (c_1, \cdots, c_k)$ with $c_i \in C$ for $i, k \in \mathbb{N}, 1 \leq i \leq k$.

Position and length of an element: the position of the first character of a model element elm within a file is defined by $pos_{seg}(elm)$. The length of elm is defined by $len_{seg}(elm)$.

Position and length of an attribute: the position of the first character of an attribute a of a model element elm is defined by $pos_{attr}(a, elm)$. The length of a is defined by $len_{attr}(a, elm)$.

Template: a template for an element elm of type t is denoted by

$$temp_t(attr_{name_1}(elm), \cdots, attr_{name_n}(elm)) := (c_1, \cdots, c_k)$$

with $c_i \in C$ for $k, i \in \mathbb{N}, 1 \leq i \leq k$. The attributes are used for the instantiation of the variables occurring in the template.

Furthermore, we define two functions that ease the formulas for deleting and inserting multiple characters:

$$\delta^{*+}(T, (c_1, \cdots, c_k), n) := \delta^{+}_{M_T}(\cdots \delta^{+}_{MT}(T, c_k, n+k) \cdots, c_1, n) \tag{3.20}$$

$$\delta^{*-}(T, n, k) := \delta^{-}_{M_T}(\cdots \delta^{-}_{M_T}(T, c_{n+k}, n+k) \cdots, c_n, n) \tag{3.21}$$

While the former inserts a tuple of characters starting from position n into a file, the later deletes the characters $c_n, ..., c_{n+k}$ at the positions $n, ..., n+k$ from a file.

As the transformation of model to source code changes is highly dependent on the type of the updated model elements, the concept for synchronization is shown exemplary for *Events* of the example model described in Fig 3.17.

A valid $Event$ element has two edges e and e' that connect it to an *HTML Element* h and to a *Function* element f, respectively. Then, a newly created $Event$ element is transformed to source code changes by

$$\delta^{+}_{t}(S_i) \rightarrow \delta^{*+}(T_{2i}, t_{event}, pos_{events}) \tag{3.22}$$

where pos_{events} references the position in the source code that contains all events and t_{event} is the following template:

$$t_{event} := temp_t(attr_{\text{`}name\text{'}}(event), attr_{\text{`}cause\text{'}}(event), attr_{\text{`}name\text{'}}(f), attr_{\text{`}id\text{'}}(h)) \tag{3.23}$$

Thereby, a new source code artifact representing the $Event$ element is inserted into the source code. The deletion of an $Event$ element is transformed as follows:

$$\delta^{-}_{t}(S_i) \rightarrow \delta^{*-}(T_{2i}, pos_{seg}(event), len_{seg}(event)) \tag{3.24}$$

Thereby, the code artifact of the deleted $Event$ element is removed from the source code. Updating a value of an attribute a is transformed to source code changes as follows:

$$\begin{aligned} \delta^{attr}_{a,event,v}(S_i) \rightarrow \delta^{*+}(\delta^{*-}(T_{2i}, pos_{attr}(a, event), len_{attr}(a, event)), \\ v, pos_{attr}(a, event)) \end{aligned} \tag{3.25}$$

Thus, the old value of the attribute is first deleted from the source code and its new value is

inserted at the same position.

As we are only considering valid models and according to our model, an *Event* element needs two edges, we can assume that for each deletion of an edge there is also an insertion of a new edge. Therefore, we only transform edge updates. Since an $Event$ element has two edges, we need to differentiate:

(a) If the current *HTML Element* h connected to an *Event* is changed to another *HTML Element* h', the transformation from model updates to source code changes is defined as:

$$\delta^{+}_{e,event,h'}(\delta^{-}_{e,event,h}(S_i)) \rightarrow \delta^{*+}(\delta^{*-}(T_{2i}, pos_{attr}(`id`, h), len_{attr}(`id`, h)), attr_{`id`}(h'), pos_{attr}(`id`, h)) \tag{3.26}$$

(b) If the current *Function* element f of an *Event* is changed to another *Function* element f', the source code is modified according to:

$$\delta^{+}_{e,event,f'}(\delta^{-}_{e,event,f}(S_i)) \rightarrow \delta^{*+}(\delta^{*-}(T_{2i}, pos_{attr}(`name`, f), len_{attr}(`name`, f)), attr_{`name`}(f'), pos_{attr}(`name`, f)) \tag{3.27}$$

We have shown above our approach for obtaining the synchronization between model and code using a trace-based approach. The live synchronization reenforces the cycle between modeling and coding phases. By incorporating the live code refinements and live preview in our process, we obtain a flexible, integrated Web development approach. The cyclic concept allows for a seamless and flexible Web CIS development, as multiple stakeholders can collaborate simultaneously on various parts and stages of the engineering process (such as design, model, code refinements).

3.4 Summary

This chapter answers the first research question of this dissertation, proposing a cyclic and collaborative approach for CIS development. Both the conceptual approach and its technical realization depend very much on the interplay between NRT collaboration during modeling and coding phases, as well as in creating suitable shared editing support within the developed applications. In the next chapter, we propose an algorithm and describe the software artifact implementation for NRT shared editing and collaboration on the Web. We propose a solution that can support the CAE HTML element collaboration and NRT communication and collaboration message exchange, that can be easily embedded into single-user Web applications in order to make them collaborative and that can support complex data types, such as model graphs. This represents the building block for our NRT shared editing infrastructure for CoPs.

We have developed our methodology based on empirical studies performed with communities and developers using widget-based Web applications. However, the approach should allow for a higher abstraction, in order to not be tightly coupled to only a single application metamodel and allow for flexibility at this level as well. Therefore, considering the NRT collaboration solution presented in the next chapter, we then present the NRT collaborative conceptual modeling approach and its implementation. Being domain-independent, metamodel based and suitable for arbitrary modeling languages, we explore how shared model editors can be automatically instantiated from metamodels, how views and guidance can be used during the modeling processes and how the technical realization is achieved. CAE MDWE modeling part is instantiated and its technicalities realized as a special use case of collaborative view-based modeling, as it will be presented later in this work.

When the snows fall and the white winds blow, the lone wolf dies but the pack survives

George R.R. Martin (born 1948)

Chapter 4

Near Real-time Collaborative Editing on Shared Data Types

Summary This chapter introduces a novel approach for NRT shared editing for multiple data types, designed for supporting collaboration on the Web. It focuses on scalability in both client-server and peer-to-peer settings and on providing an easy and flexible way to embed NRT collaboration into existing and new Web applications. The algorithm works in a decentralized approach and can be used directly for common data type and formats such as JSON, XML Document Object Model (DOM), Objects, Lists, Maps. The artifact realization fulfils the requirements for NRT shared editing in CIS and can be used as a platform for achieving collaboration for our MDWE approach.
Contribution RQ2. *Keywords:* NRT shared editing; CRDT; Collaborative editing; Conflict resolution; Intention preservation; Offline editing; YATA; Yjs. The results presented here are the main part of the CSCW aspects of this dissertation and have been published in [NJDK15, NJDK16]. This chapter contains sections with content extracted from these publications.

As seen in Fig. 5.1 this chapter presents the iteration concerning NRT shared editing technological support from our design science approach.

4.1 Problem Identification and Solution Objectives

Responding to the second research question (RQ2) of this dissertation, this chapter presents the shared editing algorithm and its library implementation, designed for existing and new Web frameworks, that support the NRT collaboration beyond simple text and data types

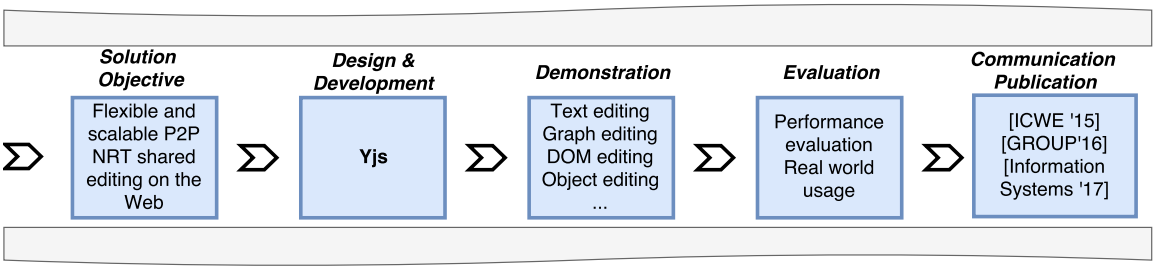


Figure 4.1: Design science approach: Yjs design and development

with linear addressing spaces. In Chapter 2, we have presented the main approaches and tools for shared editing.

As part of our design science methodology, the various iterations of our CIS support tools experiment with both OT and CRDT shared editing algorithms and frameworks. The NRT collaborative modeling approach presented in this dissertation uses in its first iterations an open source OT algorithm implementation for handling concurrency control and maintaining consistency of IMS design models and domain independent visual models. These iterations are presented in detail in Chapter 5. The OT implementation helps in the investigation and comparison of advantages and disadvantages of integrating and building complex shared editing Web applications. Namely, the prototypes realized in this work cope with a series of challenges when using OT shared editing algorithms in decentralized settings. Especially for open source and new CIS that choose an OT-based solution, drawbacks exist such as a poor offline editing support, high implementation efforts, correctness and scalability issues in P2P environments. Most OT frameworks support collaboration on text, JSON and/or XML [IN08, Ger06, HGSG12]. In decentralized settings, they require high development efforts, needed for the linearisation of all used data and a relatively high network flooding with synchronization or acknowledgement messages. Due to the difficulty of dealing with known OT collaboration puzzles, they can fail to scale with the number of users in such environments [WUM09]. In our studies, usually performed with up to three participants, the consistency was preserved and modelers could perform their tasks collaboratively. However, some copies were out of synchronization during joining or leaving and re-entering joint collaboration sessions. We identify as main reason the lack of flexibility of open source OT libraries (allowing decentralized, Web-based collaboration) to cope with complex network challenges and scale with the number of users and exchanged messages/conflicts occurred. In terms of applying operations on a local copy of a shared data, in a client-server approach there are n execution orders, where n is the number of users, whereas in a P2P approach, the clients have one connection to every collaborator and there are $n(n-1)!$ execution orders that need to be considered.

On the other hand, CRDTs enable a better scalability in decentralized environments (P2P contexts). Especially for Web applications, CRDTs are easier to implement and more flexible with respect to synchronization needs of shared objects on the Web and more

reliable. While CRDTs are designed for such settings, the choice between the two algorithm types depends much on the type of architecture needed and performance requirements. On the downside, CRDTs are rather new and few implementations for the Web exist that can easily support synchronous collaboration.

Therefore, in order to sustain our CIS infrastructure and enable NRT features for collaborative conceptual modeling, coding and easy integration of collaboration features into community Web systems, we have developed a new approach and technological support, satisfying the following core requirements:

- *Scalability* and *reliability* in P2P settings
- *Decentralized* synchronization mechanism
- *Multiple communication protocol support* such as XMPP, WebRTC, WebSockets, etc.
- *Multiple data types support* for enabling collaboration on HTML5 elements, Web pages (DOM elements) and objects, graphs (i.e., in the collaborative MDWE context), lists, XML, JSON and text
- *Offline editing* support
- *Reduced time to synchronize* shared data among clients (for example after a late join)
- *Flexibility* to customize the technology according to the community context
- *Open source* results for spreading adoption and support developer communities

4.2 Design and Development: A CRDT Approach

Our algorithmic approach[1], named YATA is created to provide a scalable solution for P2P optimistic concurrency control on the Web. The algorithm proposes a basic structure using a linked list, which can be extended to achieve collaboration on new shareable data types. YATA's linked list internal representation and a collection of predefined rules limit the number of possible conflicts and ensure intention preservation and convergence. The core idea is enforcing a total order on the shared data types. YATA also supports offline editing, being meant to cope with requirements coming from both Web and mobile clients, such as small operation updates for low bandwidth, on and off connections, random message order at receive time, etc.

YATA currently supports collaboration on linear data, trees, associative arrays and graphs. Using those types, it is possible to create more complex data types.

[1]The content of this section is a summary of [NJDK16]

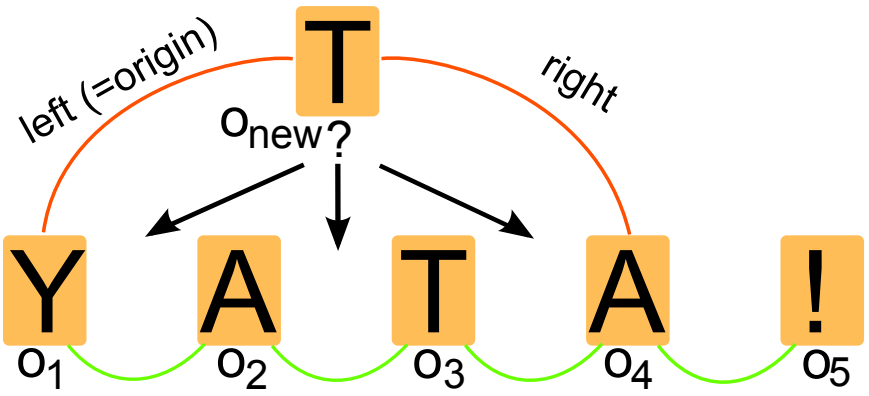


Figure 4.2: Integration example on text (linear) type.

In the following we formalize our approach and exemplify YATA'S behavior on text (linear data). After giving the assumptions, definitions and describing how convergence is achieved, we extend the linear representation to more data types and explain how those are further realized using YATA.

4.2.1 Operation Notation

Unique identifiers. Each user is represented by an unique identifier ($userid$). Additionally, each user gets an $operation\ counter$ which gets incremented every time a user creates an operation. Upon its creation, the operation thus gets assigned a unique identifier which is composed of the $userid$ and the current $operation\ counter$.

Operations. YATA represents linear data (such as text) as a doubly linked list. We define only two types of changes on this representation: *insert* and *delete*. As it is shown in Fig. 4.2, every element in the linked list is represented by an insert operation (also named *insertion*). When an insertion is deleted, it is just marked as such and not removed from the list (i.e. tombstone approach). Therefore, delete operations do not have an effect on our insert algorithm.

In Section 4.2.5 we define a garbage collection mechanism that, in combination with our insert algorithm 4.2.4, can remove deleted insertions.

We denote an insert operation as $o_k(id_k, origin_k, left_k, right_k, isDeleted_k, content_k)$, where id_k is o_k's unique identifier, $content_k$ is the content (for example a character), $isDeleted_k$ is a flag that marks an insertion as deleted, and $origin_k$, $left_k$, $right_k$ are references to other already existing insertions. We represent linear data as a doubly linked list S of insertions. Therefore, $left_k$, and $right_k$ reference to the previous node, respectively next node in the list. $origin_k$ denotes the direct predecessor at creation time (i.e., the node after which it was originally integrated to).

We define $<$ as the natural predecessor relation on S.

$$o_1 < o_2 \quad \Leftrightarrow \quad o_1 \text{ is a predecessor of } o_2 \tag{4.1}$$

$$o_1 \leq o_2 \quad \Leftrightarrow \quad o_1 < o_2 \vee o_1 \equiv o_2 \tag{4.2}$$

Example. When a user creates a new insertion at a local site, this is integrated between two insertions o_i, and o_j. The newly created insertion is therefore defined as: $o_{new}(id_k, o_i, o_i, o_j, false, content_{new})$. Note that $left_{new}$, and $right_{new}$ are defined when an insertion has been applied to the list and may change when new insertions are integrated into S, but $origin_{new}$, defined at insertion creation time is never modified. After the user integrates a new insertion at his local site he sends it (via broadcast), as is, to all users.

A special case occurs when an insertion is performed at the beginning or at the end of S, because there is no insertion to refer to as $left_*$, respectively $right_*$. This can be fixed by using special $delimiters$, which denote the beginning and the end of S, respectively. Therefore, we assume without loss of generality that an insertion always defines $origin_k$, $left_k$, and $right_k$.

4.2.2 The YATA CRDT Algorithm

The example in Fig. 4.2 shows how a received operation o_{new} is integrated in S. Here, the red connections reference the intention of the insertion, which is defined through $left_{new}$ and $right_{new}$ - i.e. insert the letter between these two letters. When the insertion is integrated, YATA assures that it will be placed somewhere between these letters. Convergence is therefore ensured, unless one or more remote operations were already inserted between $left_{new}$ and $right_{new}$, which then leads to a conflict that needs to be solved.

Definition: ***Intention Preservation***.

The intention of an insertion o_i is preserved if and only if the insertion is integrated somewhere between $left_i$ and $right_i$. This notion of intention preservation conforms to the natural perception for the intention of text insertions and it is similar to other definitions found in the literature. In [AMOI13], the intention preservation is defined when each character inserted by a user between two other characters in a document keeps its relative position between its neighbors during the editing process.

The concurrent insertion problem.

In the example in Fig. 4.2, the intention of the insertion of "T" is that should be inserted between the characters "Y" and the "A" – that is, at creation time, "T" sees only "YA". However, a letter sequence "AT" has been already inserted between these two letters. In the example, o_2, and o_3 conflict with o_{new}.

Definition: ***Conflicting insertions***.

Keeping the above notations, assuming
$S = left_{new} \cdot c_1 \cdot c_2 \cdot .. \cdot c_n \cdot right_{new}$, then we say that o_{new} conflicts with $c_1..c_n$.

In the following we define a function $<_c$ that specifies a position for o_{new} in the set of conflicting insertions $(c_1..c_n)$. As such o_{new} is integrated between c_i, and c_{i+1} when $c_i <_c o_{new} <_c c_{i+1}$. Furthermore, we show that every site converges when integrating with $<_c$ (i.e., we prove that $<_c$ is a strict total order function).

In the following we will frequently refer to the graphical representation of insertions as it is shown in Fig. 4.2. The insertion o_{new} has three references/connections. On the left hand site of o_{new} there is a $origin_{new}$ connection, and a $left_{new}$ connection to o_1. On the right hand site of o_{new} there is a $right_{new}$ connection to o_4. While $left_{new}$, and $right_{new}$ define the usual predecessor/successor relation in a linked list. The $origin_{new}$ connection will never change and is employed to find the strict total order function $<_c$.

We compose the following three rules in order to find a strict total order $<_c$ on conflicting operations.

Rule 1 We forbid crossing of $origin$ connections (red lines in the graphical representation) between conflicting insertions. This rule is easily explained using the graphical representation of insertions in the linked list. As we stated before, every insertion has an origin connection to an insertion to the left (to a predecessor). Only when two operations are concurrently inserted after the same insertion, they will have the same origin.

Figure 4.3: List visualization for the "no line crossing" rule

Fig. 4.3 illustrates the two cases that are allowed when line crossing is forbidden. Either, one operation is between the other operation and its origin, or the origin of the one operation is a successor of the other operation. Therefore, the following formula must hold for conflicting insertions o_1 and o_2:

$$o_1 <_{rule1} o_2 \Leftrightarrow o_1 < origin_2 \vee origin_2 \leq origin_1 \tag{4.3}$$

Rule 2 Specifies transitivity on $<_c$. Let $o_1 <_c o_2$. Then following rule ensures, that there is no o that is greater than o_2, but smaller than o_1, with respect to $<_c$

$$o_1 <_{rule2} o_2 \Leftrightarrow \forall o : o_2 <_c o \rightarrow o_1 \leq o \Leftrightarrow \nexists o : o_2 <_c o < o_1 \tag{4.4}$$

Rule 3 When two conflicting insertions have the same origin, the insertion with the smaller creator id is to the left. We borrow this rule from the OT approach. But in OT this rule is

$$
\begin{aligned}
& \neg(o_1 <_c o_2) \wedge \neg(o_2 <_c o_1) \\
\Leftrightarrow \quad & \neg((o_1 < origin_2 \vee origin_2 \leq origin_1) \wedge (\nexists o : o_2 <_c o < o_1) \wedge (origin_1 \equiv origin_2 \rightarrow creator_1 < creator_2)) \\
\wedge \quad & \neg((o_2 < origin_1 \vee origin_1 \leq origin_2) \wedge (\nexists o : o_1 <_c o < o_2) \wedge (origin_2 \equiv origin_1 \rightarrow creator_2 < creator_1)) \\
\Leftrightarrow \quad & (\neg(o_1 < origin_2) \wedge \neg(origin_2 \leq origin_1)) \vee (\exists o_2 <_c o < o_1) \vee (origin_1 \equiv origin_2 \wedge \neg(creator_1 < creator_2)) \\
\wedge \quad & (\neg(o_2 < origin_1) \wedge \neg(origin_1 \leq origin_2)) \vee (\exists o_1 <_c o < o_2) \vee (origin_2 \equiv origin_1 \wedge \neg(creator_2 < creator_1)) \\
\Leftrightarrow \quad & (\neg(o_1 < origin_2) \wedge \neg(origin_2 \leq origin_1)) \vee (\exists o_2 < o < o_1) \vee (origin_1 \equiv origin_2 \wedge \neg(creator_1 < creator_2)) \\
\wedge \quad & (\neg(o_2 < origin_1) \wedge \neg(origin_1 \leq origin_2)) \vee (\exists o_1 < o < o_2) \vee (origin_2 \equiv origin_1 \wedge \neg(creator_2 < creator_1))
\end{aligned}
$$

Figure 4.4: Total order equation: derivation of totality using equation 4.6

applied when the position parameters are equal.

$$o_1 <_{rule3} o_2 \Leftrightarrow origin_1 \equiv origin_2 \rightarrow creator_1 < creator_2 \tag{4.5}$$

We get retrieve the total order function $<_c$ by enforcing all three rules:

$$
\begin{aligned}
o_1 <_c o_2 \quad & \Longleftrightarrow \quad o_1 <_{rule1} o_2 \wedge o_1 <_{rule2} o_2 \wedge o_1 <_{rule3} o_2 \\
& \Leftrightarrow \quad o_1 < origin_2 \vee origin_2 \leq origin_1 \\
& \wedge \quad \nexists o : o_2 <_c o < o_1 \\
& \wedge \quad origin_1 \equiv origin_2 \rightarrow creator_1 < creator_2 \\
o_1 \leq_c o_2 \quad & \Leftrightarrow \quad o_1 <_c o_2 \vee o_1 \equiv o_2
\end{aligned} \tag{4.6}
$$

4.2.3 Algorithm Correctness

$<_c$ only depends on the $origin_*$ connection, and we specified above, that $origin_*$ never changes. We can conclude that whenever two sites compare conflicting insertions, they will find the same order for insertions. Furthermore, this implies that all sites will eventually converge. Finally, we prove that $<_c$ is a strict total order function, i.e. $\leq_c$ is a total order on conflicting operations. Therefore, we have to show that for all conflicting insertions o_1, o_2, and o_3 the ordering function $\leq_c$ is *antisymmetric*, *transitive*, and *total*.

$$o_1 \leq_c o_2 \wedge o_2 \leq_c o_1 \quad \Rightarrow \quad o_1 \equiv o_2 \text{ (antisymmetry)} \tag{4.7}$$

$$o_1 \leq_c o_2 \wedge o_2 \leq_c o_3 \quad \Rightarrow \quad o_1 \leq_c o_3 \text{ (transitivity)} \tag{4.8}$$

$$o_1 \leq_c o_2 \quad \vee \quad o_2 \leq_c o_1 \text{ (totality)} \tag{4.9}$$

Antisymmetry. Let o_1, and o_2 be insertions, with $o_1 \leq_c o_2 \wedge o_2 \leq_c o_1$.

Case 1: ($origin_1 \equiv origin_2$):

$$
\begin{aligned}
& o_1 \leq_c o_2 \wedge o_2 \leq_c o_1 \\
& \overset{4.6}{\Rightarrow} (o_1 <_c o_2 \wedge o_2 <_c o_1) \vee o_1 \equiv o_2 \\
& \overset{\text{Rule 3}}{\Rightarrow} (creator_1 < creator_2 \wedge creator_2 < creator_1) \vee o_1 \equiv o_2 \\
& \Leftrightarrow o_1 \equiv o_2
\end{aligned}
$$

The reasoning here is that the user id has a total ordering.

Case 2: ($origin_1 < origin_2$):

$$
\begin{aligned}
& o_1 \leq_c o_2 \wedge o_2 \leq_c o_1 \\
& \Leftrightarrow (o_1 \leq o_2 \wedge o_2 \leq o_1) \wedge (o_1 \leq_c o_2 \wedge o_2 \leq_c o_1) \\
& \Leftrightarrow (o_1 \leq o_2 \wedge o_2 \leq o_1) \wedge (o_1 \equiv o_2 \vee (o_1 <_c o_2 \wedge o_2 <_c o_1)) \\
& \overset{\text{Rule 1}}{\Rightarrow} o_1 \leq o_2 \wedge o_2 \leq o_1 \\
& \wedge o_1 \equiv o_2 \vee ((o_1 < origin_2 \vee \underbrace{origin_2 \leq origin_1}_{false}) \\
& \wedge (o_2 < origin_1 \vee \underbrace{origin_1 \leq origin_2}_{true})) \\
& \Rightarrow o_1 \leq o_2 \wedge o_2 \leq o_1 \wedge (o_1 \equiv o_2 \vee o_1 < origin_2) \\
& \Rightarrow (o_1 \leq o_2 \wedge o_2 \leq o_1 < origin_2) \vee o_1 \equiv o_2 \\
& \overset{origin_2 \overset{!}{<} o_2}{\Rightarrow} o_1 \equiv o_2
\end{aligned}
$$

Case 3: ($origin_2 < origin_1$): Similar to Case 2.

Transitivity. Let o_1, o_2, and o_3 be insertions, with $o_1 \leq_c o_2 \wedge o_2 \leq_c o_3$, o_3 conflicts with o_1, and w.l.o.g. $o_1 \neq o_2 \neq o_3$.

$$
\begin{aligned}
& o_1 <_c o_2 \land o_2 <_c o_3 \\
\overset{\text{Rule 2}}{\Rightarrow} & (\forall o : o_2 <_c o \rightarrow o_1 \leq o) \land (o_2 <_c o_3) \\
\Rightarrow & (o_2 <_c o_3 \rightarrow o_1 \leq o_3) \land (o_2 <_c o_3) \\
\Rightarrow & o_1 < o_3 \\
\Leftrightarrow & o_1 <_c o_3
\end{aligned}
$$

Totality. Let o_1, o_2 be insertions, with $o_1 \neq o_2$.

Totality is fulfilled if the following statement holds:

$$o_1 <_c o_2 \lor o_2 <_c o_1 \Leftarrow true$$

if and only if

$$\neg(o_1 <_c o_2 \lor o_2 <_c o_1) \Rightarrow false$$

When we apply the ordering relation (4.6) we get the formula in Fig. 4.4 and show for each case that totality is fulfilled.

Case 1 ($origin_1 \equiv origin_2$ and $creator_1 < creator_2$):

$$
\begin{aligned}
& \text{First steps are depicted in Fig. 4.4} \\
\Rightarrow & \exists o_2 < o < o_1 \\
\Rightarrow & o_2 < o_1 \\
\overset{\text{Rule 3}}{\Rightarrow} & false
\end{aligned}
$$

Case 2 ($origin_1 \equiv origin_2$ and $creator_2 > creator_1$): Similar to case 1.

Case 3 ($origin_1 < origin_2$):

First steps are depicted in Fig. 4.4

$$\Rightarrow (\neg(o_1 < origin_2) \vee (\exists o : o_2 < o < o_1))$$

$$\wedge (\exists o : o_1 < o < o_2)$$

$$\Leftrightarrow (\neg(o_1 < origin_2) \wedge (\exists o : o_1 < o < o_2))$$

$$\vee (\exists o : o_2 < o < o_1 \wedge \exists o : o_1 < o < o_2)$$

$$\Rightarrow (\neg(o_1 < origin_2) \wedge (\exists o : o_1 < o < o_2))$$

$$\vee (o_2 < o_1 \wedge o_1 < o_2)$$

$$\overset{\text{antisymmetry}}{\Rightarrow} \neg(o_1 < origin_2) \wedge (\exists o : o_1 < o < o_2)$$

$$\Rightarrow origin_2 \leq o_1 \wedge o_1 < o_2$$

$$\overset{\text{assumption}}{\Rightarrow} origin_1 < origin_2 \leq o_1 \wedge o_1 < o_2$$

$$\Rightarrow origin_1 < origin_2 \leq o_1 < o_2$$

$$\overset{o_1 \text{ and } o_2 \text{ conflict}}{\Leftrightarrow} origin_1 < origin_2 < o_1 < o_2$$

$$\overset{\text{Rule 1}}{\Rightarrow} false$$

Case 4 ($origin_2 < origin_1$): Similar to Case 3.

4.2.4 Insert Algorithm

Previously, we proved that there exists a total order relation on conflicting insertions. In this section we show how we can compute the new position for an insertion, when we already have an ordered list of insertions.

Listing 4.2.4 shows how the conflicting insertion can be solved algorithmically. The algorithm exploits property (4.3) (no line crossing) as a breaking condition. Therefore, we stop computing when origin connections definitely will cross.

The worst case time complexity of the algorithm is $O(|C|^2)$ where $|C|$ is the number of conflicting operations. In the case that the breaking condition is reached in the first iteration, no positions are compared. This is why the best case time complexity is $O(1)$. A complexity analysis is presented in Section 4.2.7.

```
// Insert 'i' in a list of
// conflicting operations 'ops'.
insert(i, ops){
  i.position = ops[0].position
  for o in ops do
    // Rule 2:
    // Search for the last operation
    // that is to the left of i.
    if (o < i.origin
        OR i.origin <= o.origin)
      AND (o.origin != i.origin
        OR o.creator < i.creator) do
      // rule 1 and 3:
      // If this formula is fulfilled,
      // i is a successor of o.
      i.position = o.position + 1
    else do
      if i.origin > o.origin do
        // Breaking condition,
        // Rule 1 is no longer satisfied
        // since otherwise origin
        // connections would cross.
        break
}
```

4.2.5 Garbage Collection

In the literature, garbage collection has been also proposed in [PMSL09] where "cold" areas of a document are identified or in Logoot [WUM09], which uses a graveyard for removed operations. Conceptually, an insertion marked for deletion can be garbage collected when all sites received the remove operation and have in their internal representation the operation that is to be garbage collected. However, it is hard to determine if all collaborators know simultaneously that a content was deleted. In the current approach, the problem is simplified by assuming that all users retrieved a certain remove operation after a fixed time period t which can be set according to the expected protocol and network characteristics (for example 30 s). In practice, YATA uses two buffers for garbage collection, to ensure that list elements are not directly removed. As such, once o_k can be garbage collected, it will be moved into the first buffer. If nothing changes, after t seconds it will be copied into the second array and from here will be removed by the garbage collector (i.e., can be safely removed from the list and the buffer). From our practical experiences and the use in production, such a delay is sufficient to ensure that content will be removed safely, without any losses that may lead to inconsistencies. This is in line with experiments performed for assessing the NRT criteria and measuring the time in which operations are being applied. These experiments (cf. Fig. 4.10) show that the average time for receiving and applying a remove operation using text (with a length under 10^3 characters) at a remote site is approx. 12 ms. Under the same conditions, receiving and applying a single remove operation with a length of 10^5 characters at a remote site is done within an average time of 39.3 ms (SD 2.45 ms), from a pool of ten measurements.

As a consequence of YATA's rules, in some cases it is not possible to remove insert operations. The reason is that for an operation that is inserted between two undeleted insert-type operations, this could lead to a deleted predecessor or successor (cf. Fig. 4.5).

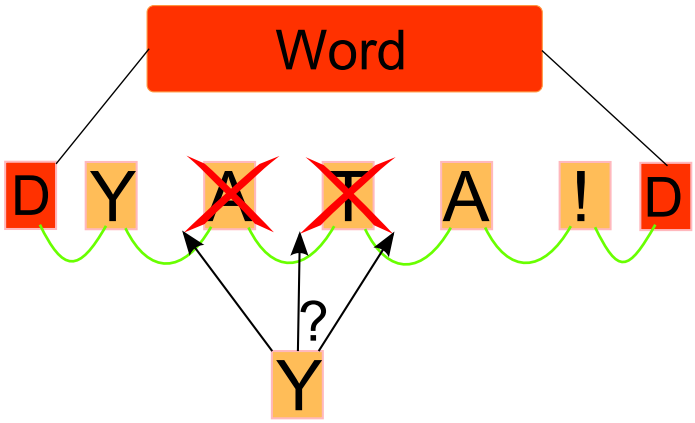


Figure 4.5: Garbage collection: removing insertion between deleted characters

In order to ensure consistency, YATA demands that a new insertion is always inserted between the most left non-deleted character and its direct successor. Only then, the garbage collector can remove all operations that are to the right of the first deleted insertion.

Furthermore, due to its design, the garbage collector in YATA may break late join mechanisms. This is because when a user is offline for a period longer than t seconds, it will still hold references to deleted operations, while online users who already performed certain removals do not. Therefore, YATA does not support garbage collection for a site while it is offline.

4.2.6 Offline Editing Support

YATA supports offline editing using the internal data representation which is maintained at each client. Once clients are online, YATA performs a check for diverged states of the shared data and synchronizes it.

Every site holds a state vector. It saves the next expected operation id per user. As an example, consider *user1* with $userid$ 1 is in a session with *user2* with $userid$ 2. Both users created two operations. As we explained above, the operation id is defined as a tuple of $userid$ and $operationcounter$. Therefore, the state vector is expressed as:
$[(1, 2), (2, 2)]$ (assuming we start counting with 0).

For synchronization, the state vector is not sent with each operation, but it is sent only once to all clients. A user that receives a state vector compares it with the local state vector and sends all remaining operations to the synchronizing client. In order to make operations integrable on the remote instances, operations are sent in the order and the form in which

they were created. Our YATA's implementation can transform integrated operations to their original form.

4.2.7 Complexity Analysis

The complexity of YATA depends on the retrieval efficiency of insertions. As such, the retrieval of a specific operation can be efficiently implemented using a balanced tree, which exposes time complexities of $O(log(H))$ for insertion and retrieval. Here, H denotes the number of all applied operations in the balanced tree implementation. The best case time complexity for applying an operation is $O(log(H))$ - i.e. there are no conflicting operations. The worst case time complexity is $O(log(H) + C^2)$ and only depends on the amount of conflicting operations (C). A comparison with other CRDTs performance, adapted from [ANIO+11] is depicted in Table 4.1. In this comparison, H the total number of operations that affect a shared document.

CRDT	LOCAL		REMOTE	
	INS	DEL	INS	DEL
WooT	$O(H^3)$	$O(H)$	$O(H^3)$	$O(H)$
WooTO	$O(H^2)$	$O(H)$	$O(H^2)$	$O(H)$
Logoot	$O(H)$	$O(1)$	$O(H \cdot log(H))$	$O(H \cdot log(H))$
RGA	$O(H)$	$O(H)$	$O(H)$	$O(log(H))$
YATA	$O(log(H))$	$O(log(H))$	$O(H^2)$	$O(log(H))$

Table 4.1: Worst case time-complexity analysis, adapted from [ANIO+11]

Discussion As it can be observed, compared to classical CRDT tombstone approaches (such as WooT or WooTO), YATA has a better time complexity. As expected, non-tombstone approaches (such as Logoot of RGA) output a better time complexity for some operation types. Regardless of the complexity, YATA offers an approach for client-side conflict resolution, which is not common among existing CRDTs. It proposes a simple data type that can be used to compose more complex ones (cf. Section 4.2.8). In Section 4.3, we show that YATA's implementation fulfills NRT scenarios and give an overview on its usage performance.

A downside of the YATA approach is that when each character is represented as an operation, the space complexity is $O(|D|)$, where $|D|$ is the size of the shared document. In OT for example, it is possible to have an empty history buffer, when garbage collection is enabled.

However, YATA outputs the following advantages over OT approaches:

- Time-complexity: YATA reduces time to synchronize, when late join is supported.

- The size of propagated messages is small: there are no state vectors that need to be propagated with each operation. OT approaches for handling tree-like data require the user to send a path-vector with each operation. This usually has the length of the depth of the changed element, which is not necessary in YATA.
- It is possible to define many data types.

4.2.8 Extendable Types

This section describes some of the basic operation types and the general data structures which YATA supports. Building on top of the data structures, one can implement certain abstract data types and thus enable collaboration on common data formats such as JSON and XML. The supported types currently include linear data types (arrays, linked lists, sorted arrays, bitmaps, etc.), trees, graphs and associative arrays.

List Manager Operation

A List Manager (cf. Fig. 4.6) is an abstract operation type that manages insert operations. It basically handles two delimiters that denote the beginning and the end of the list, as exemplified in the algorithm's description. Hence, new insertions are placed somewhere between these delimiters according to YATA's rules.

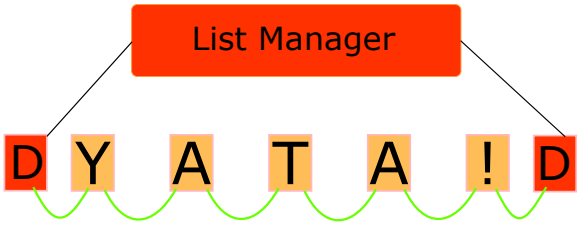


Figure 4.6: Illustration of the List operation type

The List Manager operation also handles how to address the elements in the associative list and how to transform it to a certain data type (such as String). It represents linear data structures such as lists and arrays, but it can also be used in order to represent tree-like data structures. In this case, the trees are achieved by allowing the content of insertions to contain in their turn List Managers.

Replace Manager Operation

YATA supports only insert and delete operations. However, when dealing with more complex types, update operations are also required in order to ease the development. As such, YATA supports updates of existing content by offering a dedicated type which enables content

replacement. A Replace Manager (cf. Fig.4.7 handles the replace functionality. As a basic example, consider the case where two users (with user ids 1 and 2) concurrently replace the number 0 in a text with their respective user id. In order to keep consistency, each site should reflect the replace operation and reach the same content, i.e. either 1 or 2 will replace the old number 0. YATA solves this problem by transforming it into an already solved problem, using data types which ensure consistency.

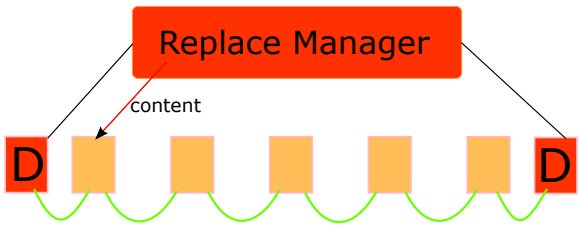


Figure 4.7: Illustration of the Replace Manager operation type

The Replace Manager inherits its functionality from the List Manager. Using this data representation, the first insertion (which must not be a Delimiter) of a Replace Manager denotes the actual content. In consequence, in order to replace this content once a user performs a new insertion, the new content will be added as the first insertion of the Replace Manager. In Fig. 4.7 the red line references the current content of the Replace Manager. Since YATA ensures that all sites will have the same content in the end, all sites eventually reference the same insertion as the content of the Replace Manager. Moreover, in order to free up memory all other insertions can be implicitly deleted.

Map Manager Operation

A map is also known as a dictionary, or an associative array. It is an abstract data type that maps $keys$ to $values$.

Fig. 4.8 depicts YATA's representation of a Map Manager operation. In order to support concurrent actions on a shared map, we assign each key to a Replace Manager. Therefore, the values addressed by keys will converge upon concurrent changes performed by multiple users i.e., users collaboratively edit the content addressed by a specific key. We can use any map data structure to map $keys$ to Replace Managers. The values can contain primitive data types or YATA own types. This construct is suitable for easily enabling collaboration on name/value pairs (objects, dictionaries, etc.). The current $value$ of a key is replaced and retrieved by accessing the respective Replace Manager.

Representation of Specific Data Formats

Using YATA, by combining the simple types explained above, data formats such as JSON or XML can be realized as shared data types.

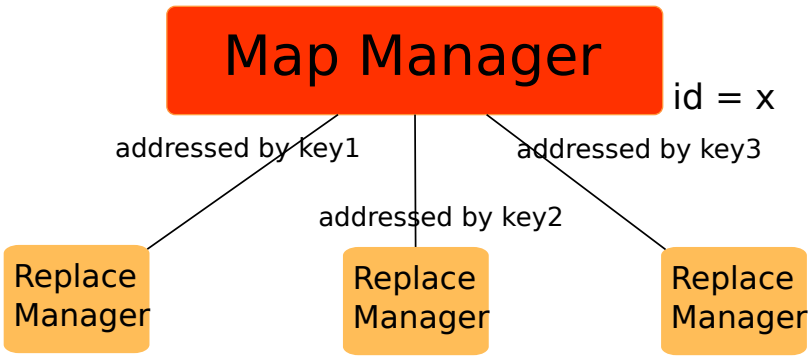


Figure 4.8: Illustration of the Map Manager operation type

The JSON data format is built on collections of name/value pairs and ordered lists of values. Hence, with YATA JSON can be easily created by using a Map Manager, in combination with other data types which can be used for the attributes, such as further Map Manager operations or List operations (for attributes implementing strings or arrays).

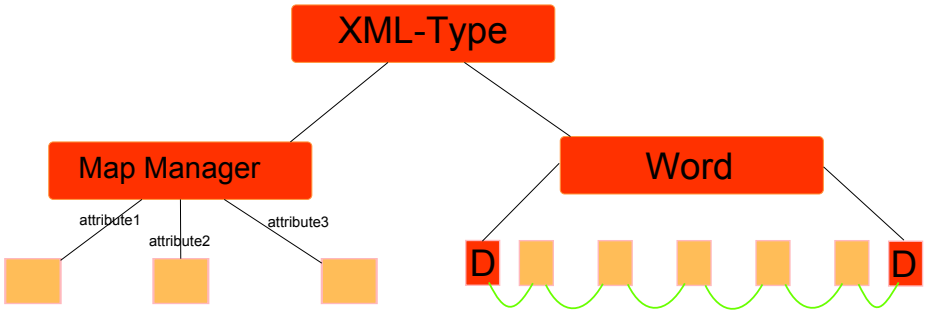


Figure 4.9: Illustration of XML as an operation type

YATA can also enable NRT collaboration on XML (cf. Fig. 4.9). An XML DOM Element[2] has XML attributes and an ordered list of children (XML elements or XML text). Therefore, we compose an XML-Type in YATA as a Map Manager (which handles the XML attributes) and a List Manager (which handles the children). However, since the List Manager and the Map Manager support to insert any value, we have to put some restrictions to the structure of the XML-Type:

- The Map Manager, which handles the XML attributes, must only map from a non-empty string to a non-empty string
- The List Manager, which handles the children, must only contain other XML types, or strings which represent XML text elements

[2] `http://www.w3.org/TR/REC-xml/`

4.2.9 Yjs: The P2P Shared Editing Framework

We have implemented the YATA approach as an open source JavaScript library, named Yjs [NJDK15]. In contrast to similar OT, CRDT and shared editing implementations that support only a very limited number of document structures in Web-based settings or represent them differently, the Yjs framework encourages developers to build custom data types. A custom type can use existing implemented types (as previously explained) in order to give meaning to the actions on the data and to fire custom events. Yjs has implemented support for *list*, *associative arrays*, *XML*, *text*, and *rich text* types. Yjs works on modern Web browsers, including mobile (on Android devices) and offers a quick and easy way to embed collaboration in Web applications.

In order to maintain modularity and to be able to employ Yjs in various Web engineering settings, the communication protocols and the shared data type formats support are implemented as dedicated interchangeable modules. This greatly simplifies the process of integrating the framework into an existing project, since such projects typically use diverse communication protocols (WebRTC, Web Sockets, XMPP), and collaborate on various data types and data formats (such as XML, JSON, graphs). The support for the different communication protocols is implemented in connector modules. The collection of connector modules and type modules are available as open source JavaScript libraries on GitHub[3]. Currently, connectors are implemented for WebRTC, Web Sockets and XMPP.

4.3 Performance Evaluation

Apart from the correctness reasoning explained above, we have evaluated the approach and the Yjs library in terms of responsiveness - how quickly an operation is integrated into each user's local structure and scalability - what is the impact of the number of users in this setting [NJDK15].

In order to evaluate the speed and correctness of YATA we performed multiple automatic tests simulating many users working exhaustively on a single shared document. The simulation introduced certain constraints: generation of random operations, different arrival orders of operations at different sites and network delay with operations which were not ready to be applied because they were dependent on other pending operations. As our implementation is meant for client-side applications and runs on the Web, we could not use existing benchmarking tools described in the literature [ANIO+11]. Moreover, using collaboration logs from online text editing systems such as Wikipedia would have lead to very few conflicts on the shared document. Instead, we chose an alternative approach with the goal to measure the fulfillment of the NRT scenario in distributed settings. We first used the testing environment with a test connector to measure the scaling capability with the number of operations by measuring how many concurrent operations YATA can

[3] `https://github.com/y-js`

apply locally. Second, we measured the time performance for applying insert and delete operations between two Yjs instances, including the network delay using the Web Sockets connector. Similar to [ANIO+11], we consider that the framework performs in NRT if the response time for the insert operation is in average under 50ms.

The testing environment can be configured with respect to number of users and number of actions that are created. Furthermore, it is possible to configure the test environment in such a way that the collaboration is restricted to a specific data type (text only or text and JSON with primitive data types). The users are collaborating concurrently with each other, whereby each simulated user can perform one of the following actions: *wait* for a time period, *retrieve an operation* from a random collaborator, *go offline* (all operations that are currently sent to the users get lost. After going offline, the user is allowed to create more operations, simulating late join), *go online* (user reconnects to all users, retrieves all missing operations, and sends all offline generated operations). For text, a user can perform insert and delete actions. For JSON, the user can perform randomly the following actions: find a random child, create a new property, replace an existing property or delete a property.

When a configured amount of actions are executed, the simulated users stop generating additional actions and wait for all incoming operations to be executed. When the data types of every simulated instance converge, we consider that the test framework succeeds.

We used our test framework with the creation and execution of 10000 actions on Text and JSON with a specified number of users, ranging from one to ten. The test ran on one CPU only (Intel i7 - 3.40 GHz). All mentioned constraints were considered. We ran the test 15 times in order to get good average times. The time to create 10000 random actions and apply them to all collaborators was measured and divided by the number of users times the number of created actions: operation per millisecond $= \frac{time}{10000 \times |users|}$ The time for transforming operations did not thwart the performance noticeably. For a setting with 7 and 8 users, the test framework registered 250 operations executed in one ms, whereas for 9 and 10 users the average time was approx. 230 operations per ms. This did not even change when we restricted the test framework to work on text only, where even more conflicts should happen.

The time performance was measured using two Yjs instances using one Chrome browser on a laptop (Intel i7 - 2.8 GHz) with a wireless Eduroam connection. Using Web Sockets connector, we measured the time difference (in ms) between applying an insert or delete operation on the first instance and applying the same operation on the second instance.

This shows the delay between creating an operation on a site and seeing the effects of that change at the second site, including the delay created from the message propagation across the communication channel (i.e., Web Sockets).

The experiment (cf. Fig. 4.10) was performed with a single insert or delete operation using a text editor. Each operation was executed ten times in order to obtain a reliable estimation. We also increased the content of each operation (in characters), ranging from 1 to 10^3 characters. The results for the insert operations are represented in gray and for the delete

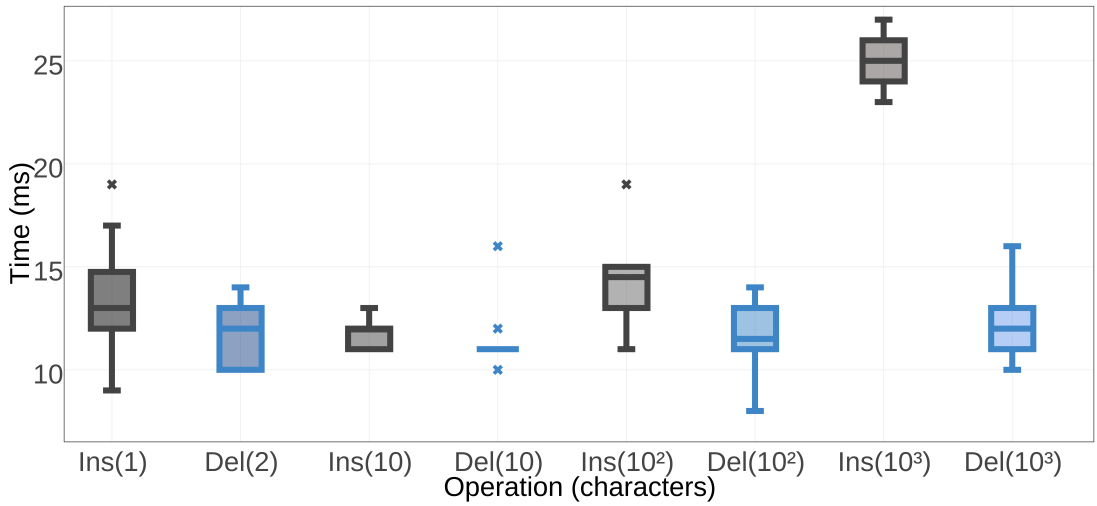


Figure 4.10: NRT performance for applying one remote operation with different content sizes

operations in blue. As it can be observed, the average time for all operations is smaller than 25 ms. We also used the same environment to write a text of 1000 characters. Here, we obtained an average execution time for an operation of 12 ms, which is consistent with the results presented in the figure for inserting and deleting one character.

Overall, the test results fulfill our expectations for the framework's performance in NRT settings.

4.4 Communication: Adoption and Lessons Learned

The Yjs library with its various connectors and types was well received by the open source developer community. The algorithm was presented at the ACM GROUP 2016 conference [NJDK16] and Yjs received the best demo and best poster awards at the International Conference on Web Engineering 2015 [NJDK15] and was also presented at the yearly FOSDEM developer conference in Bruxelles, Belgium, 2015. The Yjs library is the main shared editing driver in our approach and the proposed applications, as it will be shown in the next chapters. However, it was used also in further research projects at our department, as well as in other open source and academia projects. In a two years period, the main Website of the framework gathered more than 14000 page views from over 5000 users. Since its first release, the GitHub project has received more than 550 stars and the library build was downloaded for over 33 645 times (c.f. Fig. 4.11).

Yjs was used to enable shared editing with the popular Quill rich text editor, which is very similar to Google Docs. The richtext component is also used by companies in production,

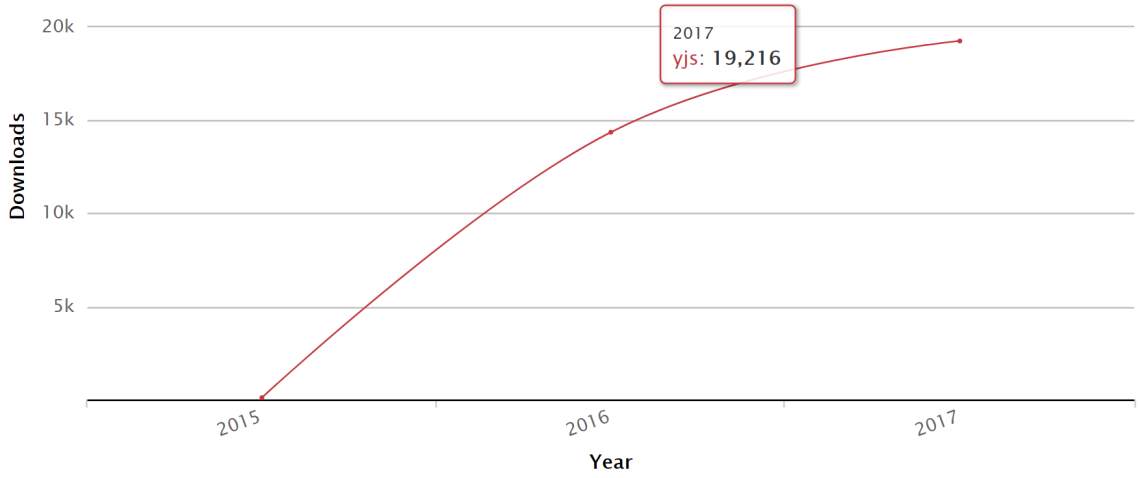


Figure 4.11: Yjs npm JavaScript package manager downloads `https://npm-stat.com`

for NRT shared text editing on the Web, embedded in WebRTC video meeting tools, etc. Furthermore, Yjs was used to create a tool for achieving liquid Web applications [GP16], by providing a seamless user experience in using different Polymer Web components distributed across various devices. The evaluation of a Yjs-based collaborative drawing tool on Web videos [KNK15] (performed using the WebRTC connector) showed that during concurrent drawings the framework proved to be reliable while keeping the NRT behavior, for all participating peers. Several other examples for manipulating text, HTML5 elements, JSON objects and additional usages in various projects are provided on the Yjs Website.

Less is more

Mies van der Rohe (1886 - 1969)

Chapter 5

Near Real-time Collaborative Modeling for Information Systems

Summary We introduce a novel approach for collaborative conceptual modeling. Based on a domain-dependent model editor used within the context of a professional CoP, it abstracts a domain-independent, visual, synchronous metamodeling concept. **Contribution** RQ3. *Keywords:* Collaborative conceptual modeling; Metamodel to model generation; Viewpoint abstraction; Domain independent visual modeling; Collaboration awareness; User nudges; SyncMeta. The results presented here have been published in [NDK13, DEN+14, DNE+15, NRD+16, NRD+18]. This chapter contains partial information and content extracted from these publications.

This chapter contains the iteration of our design science approach concerning collaborative conceptual modeling (cf. Fig. 5.1). This represents a key step in our research for studying new social means to interact with models and an important part of the realization of our CIS-centric MDWE framework presented in Chapter 3. The final realization of the SyncMeta conceptual modeling framework is built upon the Yjs library. Its design considerations after several release cycles also take into account the lessons learned and advantages of our Web-based collaborative editing studies.

The visual metamodeling approach is used for the generation of arbitrary lightweight model editors. The collaborative, NRT conceptual modeling is achieved as a result of multiple iterations, using state-of-the-art shared editing techniques, presented in previous chapters, namely 2 and 4. Although the approach is designed for usage in CoPs, the conceptual modeling approach is in itself a contribution to the information systems domain. In order to understand the requirements for achieving a general, domain-independent collaborative conceptual modeling approach, we start with a first iteration, proposing to transform the practice of learning design teacher CoPs in modeling IMS Learning Design documents. Here, using a collaborative prototype we identify the main conceptual and engineering

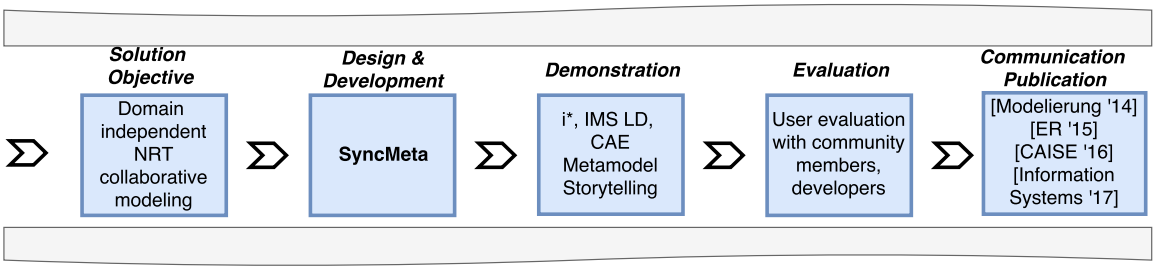


Figure 5.1: Design science approach: SyncMeta design and development

challenges for achieving NRT collaborative modeling on the Web, according to the used design science methodology. Then, we describe the SyncMeta approach and SyncMeta implementation, which fulfill our stated objectives. SyncMeta is the core artifact of the conceptual modeling contribution of this dissertation and represents the main basis for the Community Application Editor's approach and realization.

5.1 First Iteration: Domain Dependent Collaborative Modeling for Professional Learning Designer Communities

5.1.1 Problem Identification and Solution Objective

As outlined in the related work chapter, in analyzing professional learning designer CoPs it has been observed that the current IMS LD tools do not support any synchronous collaboration between multiple users, on a shared model[1]. That is, two people located at different networked sites could not collaboratively design/edit a shared IMS LD document at the same time. Existing tools were able to handle learning design sharing using import/export mechanisms only. This lead to disadvantages in the shared practice of the community, as designs were rather based on a individual author's pedagogical principles rather than reflecting collaborative ideas. The import/export of sharing existing UOL design and the use of shared repositories or electronic media file transfer took a lengthy process and was prone to duplication, since the document to be shared had to be sent to all participating authors. Moreover, incorporating all the changes made by multiple authors on an existing design was prone to errors and finally, since most of the IMS LD tools were desktop applications, they required software download and installation procedures.

We therefore present an approach for the realization of NRT collaborative IMS LD modeling on the Web. Our objectives include the reengineering of existing desktop-based, single-user applications into a Web-based application that includes a collaborative space allowing

[1]This section contains content published in [DNTK13, NDK13]

multiple users to join a collaborative modeling session and synchronously edit a shared learning design.

Therefore, we have extracted the following functional and non-functional requirements, based on the IMS LD specification and the existing tools and representations:

- *NRT communication and live propagation of updates*: distribute in NRT all create, edit and delete operations across all participating community members in a modeling session
- *Collaborative visual authoring*: activity sequences should be created graphically and the learning design generated visually while maintaining consistency
- *Direct local execution*: local operation should not be affected by network latency
- *Unconstrained collaboration*: users should be able to edit any part of the learning design at any time
- *Creation, edit and deletion functionality*: allow users to create, update and delete in a NRT collaborative fashion: new activities, properties, learner or staff roles, environments, learning objects, services, resources and role-parts
- *Export* the visual learning design
- *Workspace awareness*: provide users information about who is currently working on the shared document
- *Late join support*: allow users to join a collaboration session at any time. Synchronize the newcomers's instance of the application with the current document status.
- *NRT collaboration reusability*: reuse the NRT collaboration features in other community applications

5.1.2 Artifact Design and Development: SyncLD

The first solution we proposed for solving the lack of collaborative LD authoring is a Web-based framework named SyncLD, that supports the NRT message exchange between multiple clients and conflict resolution through the usage of a P2P OT engine. SyncLD combines features of existing IMS LD authoring interfaces and reengineers them to work on Web browsers, in a NRT collaboration environment. The activity modeling metaphors are similar to those in OpenGLM and WebCollage. The editing of detailed element properties is achieved using a familiar tree structure navigation where editing forms for the elements can be accessed. This is a common user interface metaphor in most IMS LD authoring tools. The key novelty is the Web-based NRT collaboration. IMS LD authoring can be approached very flexibly. There are only few restrictions on the order of creation of elements, which relate to

situations where an element references another element. Logically, the referenced element must be created before referencing it (for example an activity referencing an environment). Other than that authors are free to choose in which order and level of detail they want to proceed during authoring. This is one of the reasons why we chose IMS LD authoring as a pilot application for our NRT collaboration technology.

Concerning the visualization aspects of SyncLD, the interactions with the IMS model are realized through visual modeling of IMS LD compliant units of learning. In a shared space, users can concurrently work on a shared learning design model. All collaborators can view all the modifications on their editor instantly. As IMS LD is an XML-based specification, we flatten this representation to a linear data structure using unique identifiers, and synchronize it among the multiple collaborators using an OT mechanism. Fig. 5.2 presents the abstract architecture of SyncLD. It features several interoperable frontend

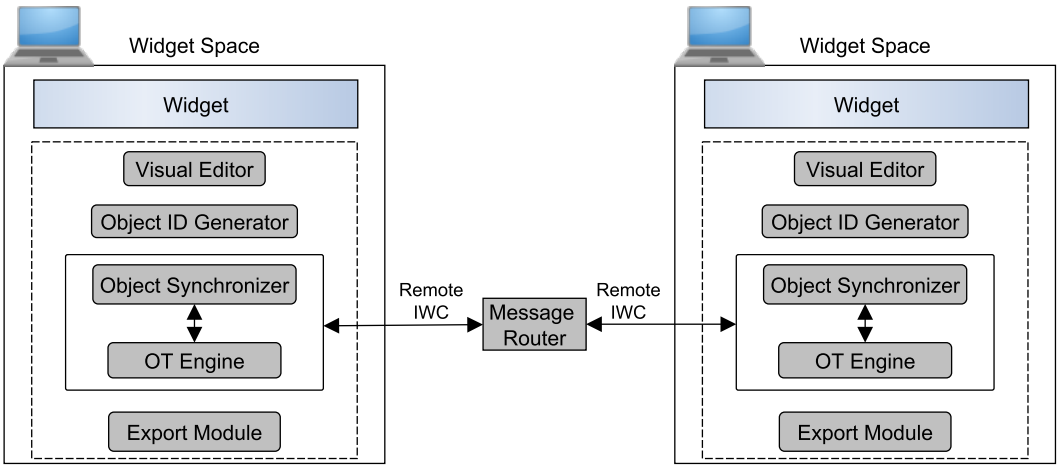


Figure 5.2: SyncLD architecture

modules. The “Visual Editor” deals with the modeling aspects at the interface level and the “Export Module” permits an IMS LD compatible export of an XML document containing the authored model, meant to be persisted or used in other authoring applications. The **Visual Editor** module is the module for the SyncLD user interface and handles the management of the various learning design authoring steps which involve the creation of objects (activities, environments, roles, etc.), deletion of objects and property value modifications. These operations can emerge both from the local and remote users. The various UI elements in the SyncLD widget are dynamically updated as a result of executing operations.

The “Object Synchronizer” and the “OT Engine” ensure that possible conflicts are resolved and that each client operates on the same version of the learning design model. The **Object Synchronizer** manages the non-conflicting part of the collaboration on the learning design models, involving object manipulation. The objects can have multiple types, such as “learning activity”, “support activity”, “role”, etc. The synchronizer deals with operations

Table 5.1: Example of conflict-prone object properties

Property name	**Property format**	**HTML representation**
Activity description	Editable text	Textarea
Prerequisite	Editable text	Textarea
Learning objectives	Editable text	Textarea
Visibility	True of false (single selection)	Checkbox
Resources	Option (single selection)	Select box
Environment	Option (multiple selection)	Multiple checkboxes

such as “create” or “delete”, which can be performed on these objects. Because we consider such operations to create unique objects, the module does not use the OT engine; instead it allows users to collaboratively create and delete such elements.

The **OT Engine** builds on OT for maintaining document consistency during concurrent edits by multiple users. The local operations on the shared model are directly reflected, thus providing a short response time. When users join an editing session, the shared authoring state is replicated at all collaborating client sites. When a user makes a modification, a message containing the necessary details is sent to all remote sites. Users can alter any part of the model during authoring and can see the modifications coming from the remote sites in NRT. The objects in the prototype have a certain number of properties. Some of them are editable by users, while some just provide predefined options. An example for the properties of an instance of a learning activity are presented in Table 5.1. Concurrent operations on any of the properties listed in the table may lead to conflicts or inconsistencies in the clients’ model representation. The OT Engine component applies OT algorithms for transforming the incoming remote events against the local client operations history and returns the new (transformed) operations. Among the actions that need the OT engine and that may lead to conflicts one can enumerate activity renaming, creation and deletion of activity connections, value selections and text editing.

We assign to each element a unique identifier, using the “Object ID Generator” module. The given identifier is unique across all participating sites. As such, we fulfill the IMS LD specification requirements, which state that each entity (element) should have an unique identifier distinguishing it from all other elements. This also enables referencing of other elements. Furthermore, during a collaborative session, when a user performs an operation on one of the objects, a message is sent to all other participating sites specifying the object the operation should be applied to. The unique string generation across all sites uses a hierarchical nested approach. Every node on a given tree structure is assigned an *id* of type string, made by concatenating the *id* of its parent node, a text that indicates the node type (“Resource”, “Environment”, “Learner”, etc.) and a number (a unique integer, generated for all the nodes in a total order). The root node’s string is concatenated from the *userId*, the node type and the unique number. Incorporating the *userId* in the pattern is crucial as it guarantees the uniqueness of objects in the case when multiple users create the same type of

objects. The *userId* is unique across all participating client sites. The unique number (index) is an integer generated by a function which analyses the sibling nodes' ids and generates the next maximum number.

Finally, the objective of the **Export Module** is to convert the current learning design model into an XML document and resource package that is conformant with the IMS LD specification. The exported file can be imported in IMS LD run-time systems or in other IMS LD authoring applications.

Implementation

To support our design decisions we used Web widgets and the open source Java-based ROLE SDK, as it provides the relevant technologies for the SyncLD requirements. SyncLD uses the widget container services for user management and the collaborative widget space management features with NRT enabled features. The widget space is the working context for users and widgets. Multiple users can collaboratively manage and interact with various widgets located in the same space. Also, the space offers the core functions for managing users' presence, multi-user chat, adding-removing widgets, etc. The NRT component is based on the Extensible Messaging and Presence Protocol (XMPP) [SA11] and uses the XMPP publish-subscribe extension (XEP-0060) [MSAM10] for realizing the inter-widget communication and the multi-user chat extension (XEP-045) [Pet08] for the communication and presence information within the space. Another basic widget-specific feature used by SyncLD is inter-widget communication (IWC) [GVD+11] Through IWC exchanges, multi-user and/or multi-widget applications can be built. SyncLD uses the remote IWC for propagating the authoring operations, updates and synchronization messages, in a multi-user NRT collaboration scenario. In collaboration with user and space management services, the platform NRT service manages one dedicated publish-subscribe channel per space for IWC including whitelist-based access control. The *IWC proxy* routes outgoing IWC messages to the affiliated XMPP server via the XMPP connection and incoming messages to all widgets in the space via HTML5 Web Messaging [Hic11]. Furthermore, the platform supports secure authentication and authorization using OpenID Connect, allowing for unique representation of the users from a certain space and therefore for identifying the users which are collaborating. In SyncLD, widgets are used for creating a dynamic authoring environment where the collaborative authoring session can be augmented with other tools (widgets), such as video conferencing, multi-user chat, shared text editing, Google Docs, and so forth.

The OT module uses the OpenCoweb open source project. It is a standalone Javascript library, which guarantees document convergence, given that all the local operations are performed and sent to the remote clients and that all the remote incoming operations are honored. Each participating site has its own instance of the OpenCoweb OT engine. The local application has to call the `OTEngine.localEvent` method for all local changes and send the object returned from this call to the remote peers unchanged. When remote

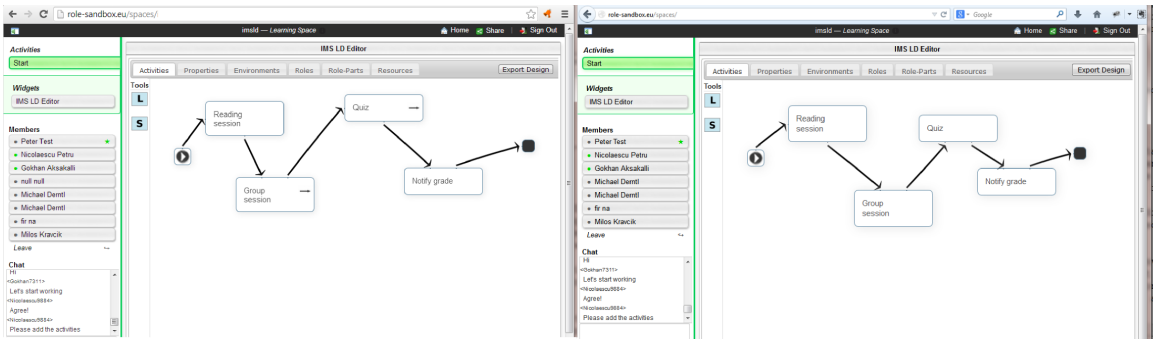


Figure 5.3: Sequence of activities in the SyncLD user interface showing synchronized model instances in two side-by-side browsers

operations are received, the engine passes them to an `OTEngine.remoteEvent` method. The transformed operation returned by this method is then applied to the local document. The OpenCoweb OT engine can be used in one application for several collaborating objects. For example, an application can have a chat and a text editor. Each one of these parts are treated separately by the engine. We consider each editable field of a given object as a separate collaborating object. Thus, whenever an edit event occurs on a particular text field, the OT engine is called by passing the combinations of the *id* of the object and the *id* of the HTML field as an argument.

At the interface level, the visual modeling is realized in two ways: a tree structure for detailed properties of elements and a UML-like activity diagram approach for producing the sequence of learning and support activities. The activities are created by dragging and dropping the `L` (learning activity) or `S` (support activity) icons from the left-hand toolbar onto the drawing canvas. The flow sequence between them is built by connecting the activities via a directed arrow. A screenshot depicting four connected activities, edited in two browser instances is presented in Fig. 5.3. Activities also have a tree structure representation that is used to provide a convenient navigation for editing the various properties of the activities. The rest of the entities, such as environments, roles, role-parts and resources are presented as a collaborative tree structure. Each entity type has a separate tree-like visualization. To modify a property a user selects a specific node from the tree.

Finally, for the export functionality, the conversion process is divided into series of tasks, such as analyzing the JSON objects in the visual learning design. This involves differentiating between attributes and child elements, eliminating empty elements from appearing in the XML document and extracting the relevant data. Another task is creating a new text file for every property that is editable (a learning activity's description, prerequisite, learning objective, etc.), which is required by the IMS LD specification. The files are represented as a separate XML element which can be referenced by other elements. Once all objects have a valid structure and are glued together accordingly, they are converted to XML elements using the function "json2xml", that takes a JSON object as an argument and returns the equivalent XML representation. This function—an efficient and precise algorithm—is

reused in our prototype with a slight modification to include namespaces [Goe06]. The resulting XML is stored as the imsmanifest file mentioned previously, and zipped along with the referenced file resources.

5.1.3 Evaluation

End-User Evaluation

Methodology: The end user evaluation was performed by providing a predefined authoring scenario, which was enacted in each evaluation session by three concurrent users with varying levels of IMS LD expertise under the guidance of a moderator, who was an IMS LD expert. The scenario consisted of four steps as listed in Table 5.2. The three authors were instructed at the beginning of each step and synchronized at the end of each step by the moderator. During the session each participant had as a backup a sheet with a description of his/her duties during the authoring session as listed in the table. The scenario was designed to allow users to experience the key features of synchronized collaborative IMS LD authoring while increasing the chance of concurrent edits. At the end of the session the users were asked to fill an online questionnaire that surveyed their perceptions of usability and usefulness of the tool in seven multiple-choice questions and open questions on aspects they liked and disliked the most about the tool, respectively.

Results: We had five evaluation sessions with three participants each (a total of 15 participants). They were asked to rate their IMS LD expertise on a Likert scale ranging from novice (1) to expert (5). The mean rating was 2.6 ($\pm$1.3), slightly below the middle point on the scale.

The rating results of the multiple-choice questions are displayed in Fig. 5.4. All but two participants agreed or strongly agreed that the tool was easy to use (mean $= 3.9$ on Q_1). In the open-ended questions some negative comments related to usability were about not being familiar with IMS LD which complicates the use of the tool, which indicates that the tool would need some in-place assistance This was marked as a key requirement to be addressed for the next iteration, where we *nudge* users what to do next at each stage. In the second question (Q_3) users indicated that they encountered some errors during use (mean $= 3.9$).

The tool did not have any visual awareness indicators to indicate to a user what part of the learning design model the other users are currently editing. However, since the scenario made them edit the same portions of the model in each step, there was some built-in awareness. This is also indicated by the responses to Q_7 which averaged a rating of 3.3. In the open questions users emphasized the need for awareness mechanisms ("it is not so clear what the others are currently doing").

We also asked whether the users felt that there was no need for an additional communication channel (such as audio, chat) while collaboratively editing the learning design (Q_6). The average rating of 3.3 along with a high standard deviation shows that the opinions differed

Step	Moderator	User 1	User 2	User 3
1	Tell the three users to perform step 1.	In the Resources tab, add a new resource '*Learning materials*'. Add a random local file from your computer to this resource and title it '*Tech-writing book*'.	In the Resources tab, add a new resource '*Conference materials*'. Add a random local file from your computer to this resource and title it '*Conference slides*'.	In the Resources tab, add a new resource '*Quiz materials*'. Add '*Online material*' to this resource, with the title '*Online quiz*' and the url '*www.example.org*'.
2	Create environment '*Study materials*'. Tell the three users to perform step 2. While they do so, create environment '*Grade report*'.	Wait for the moderator to create the '*Study materials*' environment. Then add a learning object '*Tech-writing book*' to this environment. Select learning object type: '*knowledge-object*' and resource: '*Learning materials*'.	Wait for the moderator to create the '*Study materials*' environment. Then add a learning object '*Quiz*' to this environment. Select learning object type: '*test-object*' and resource: '*Quiz materials*'	Wait for the moderator to create the '*Study materials*' environment. Then add a '*New Conference*' service to this environment with the title '*Tech-writing conference*'. Associate the '*Conference materials*' resource to this service. In the same view, please select the student and the tutor as participants, and the tutor as manager and moderator.
3	When user 3 complains that he cannot add roles to the conference service, tell user 1 and 2 to first complete step 2. Then tell user 1 and 2 that they should proceed with step 3, i.e. user 1 to add student role and user 2 to add tutor role. Ask user 3 whether he can see the roles popping up now.	Add learner role with title '*Student*'.	Add staff role with title '*Tutor*'.	Wait until roles from step 2 appear in your user interface.
4	Tell users to create learning activities in step 4 by dragging the '*L*' on the canvas. In the meantime add support activity '*Notify grade*', with environment '*Grade report*' and supported role '*Student*'. When all users ready, tell them to go to '*Properties*', activity '*Reading session*'. Modify the description to '*Read the provided materials*'. Then tell them to go to the '*Activities tab*' and wait. Tell them that you are now connecting the activities. Do it.	Create learning activity '*Reading session*'. Put this in the description: '*Read the materials*'. Select environment '*Study materials*'.	Create learning activity '*Group Discussion*'. Put this in the description: '*Technical writing*'. Select environment '*Study materials*'.	Create learning activity '*Quiz*'. Put this in the description: '*Publish quiz*'. Select environment '*Study materials*' and completion-rule '*time limit*'.

Table 5.2: End-user evaluation session script with three users collaboratively creating an IMS LD model

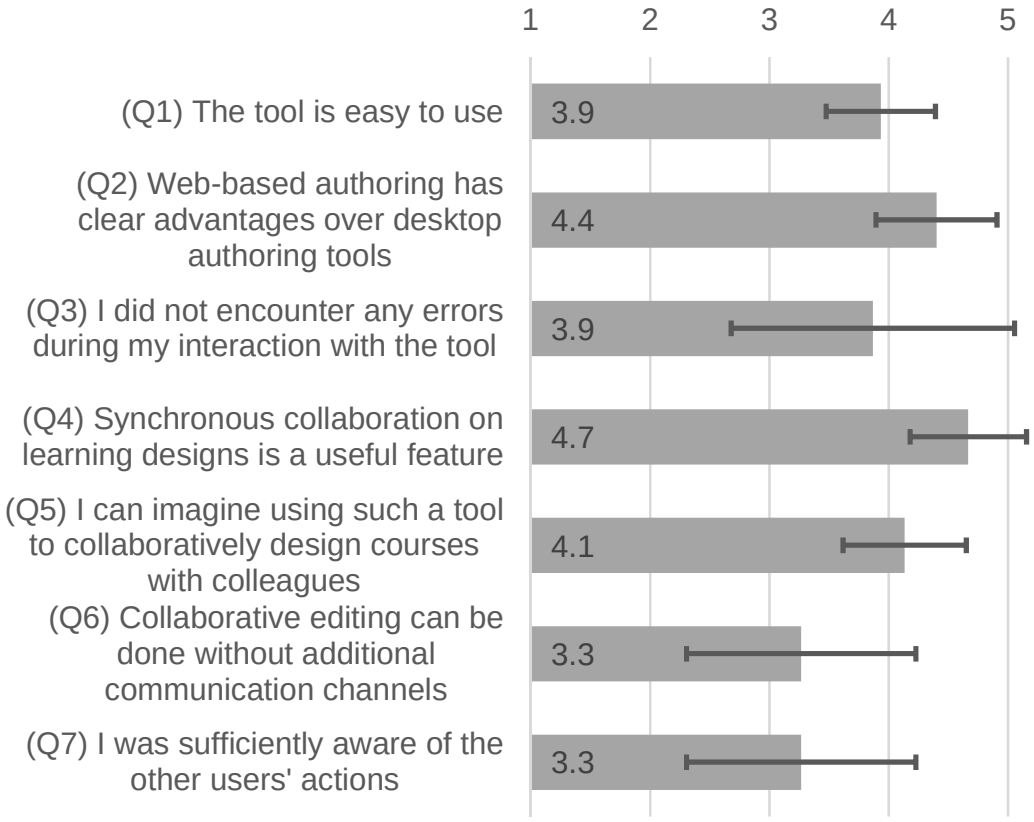


Figure 5.4: Evaluation survey results indicating average rating and standard deviation for each question [1 = fully disagree … 5 = fully agree]

considerably. Clearly, being able to speak to collaborators is an advantage, yet obviously several users were confident that they could use the tool collaboratively without additional communication channels.

All participants agreed or strongly agreed with the statement that Web-based authoring has clear advantages over the desktop authoring tools (Q_2; mean $= 4.4$). In addition, all of them agreed that the synchronous collaboration feature was useful (Q_4; mean $= 4.7$), which provide strong support for the objectives and outcome of this research. Some of comments about what they liked about the NRT editing include: “co-designing and sharing a common space– “to see other users’ changes almost in real-time– “everything is automatically synchronized/stored (like Google Docs)– “work can be done faster if everyone works in a different area– “to see things done by collaborators popping up.”

Last but not least, almost all participants agreed that they could imagine using such a tool to collaboratively design courses with their colleagues (mean $= 4.1$ for Q_5), which is an encouraging result when thinking about productive use of this or similar tools by learning designers for real tasks.

Table 5.3: Technical evaluation of messaging infrastructure and OT Engine

Test Performed	Average Value (ms)	Standard Deviation (ms)
Continuous character input	778	1237
Writing (normal speed)	457	856
Single operations	99	58
Time difference at receivers	28	24

Technical Evaluation

Parallel with the user evaluation, we considered also the evaluation of the technical realization of SyncLD, the collaboration and NRT features of the prototype and the underlying Javascript library integration. Two approaches were used: the first approach was to perform functional tests during the development of the prototype. Moreover, in the ending phase of the development cycle the performance of the XMPP-based NRT messaging infrastructure working together with the OT engine was tested. The second approach involved the investigation of the user study results. The performance evaluation of the framework was conducted using the same NRT shared editing setting used in the user evaluation. The analysis was performed using three collaborating browser instances with three different users logged in with their accounts in the same ROLE space. The clients accessed a wireless network different from the one containing the XMPP server. Each clients collaborated using a text editing widget (containing a text area HTML element connected to the OT library and the ROLE IWC for event handling). Events were keyboard-generated using the text area element. All underlying editing infrastructure messages were logged, persisting information such as the exact time of sending and receiving each message together with operation type and the ids of the sender and receiver. The experiment consisted of several sessions, in order to test the different possible IWC exchange scenarios between the clients. Two shared text editing sessions at usual writing speed were considered and one with continuous character input. Because part of SyncLD collaboration involved single operations (activity creation, checkboxes, etc.), single events (created using single character input at a time interval greater than one second) were also analyzed. In each session one hundred messages were generated. Finally, the message delay between different clients was analysed, using one sender and two receivers and measuring the time difference between the message receive time at the receivers. The results are presented in Table 5.3.

The performance evaluation results show that our methods can enable NRT collaboration for Web applications. Although not perfect, the receive time delay between the clients can be considered as a near real-time perception for remote collaborators. The increased time difference obtained for the test cases containing multiple characters input is due to the buffering of characters and the bundling of a certain number of such events under a single IWC intent, in order to minimize the network traffic. A further motive is represented by the synchronization messages and large vector clocks that are used by the OT library.

For the collaborative features, the focus was in measuring the convergence (consistency of the model at the participating clients) of the learning design when multiple users are collaboratively authoring. This was carried out as an extension of the user evaluation, where we tried to gain knowledge of the technology usability and reliability. In this sense, we used part of the users presented in the user study, in order to assess the consistency of the client replicas. After the users collaboratively designed the UOL, they were asked to export the UOL model using the tool. We collected from each user the downloaded UOL package and inspected each file in the package for convergence. Each package contained one XML ("imsmanifest") file and four text files. A comparison of the content of the files was carried out manually since, except the XML file, the content of the text files was very short (predefined by the guideline). The comparison of the XML files was performed by inspecting the elements. It was observed that the design at each of the three sites was convergent, meaning, the "imsmanifest" XML file as well as the text files generated by each user had the same content. However, we also note that there are factors which can have an impact on this result such as the overall design being guided by predefined steps, number of participants, etc. Since part of the requirements (such as the space presence signaling, the ability to share and join a space, etc.) are supported by the underlying ROLE SDK infrastructure, further performance tests were not considered as being within the scope of this study.

5.1.4 Communication: Discussion and Lessons Learned

SyncLD was presented as a demo at the 8th European Conference on Scaling up Learning for Sustained Impact (EC-TEL 2013) [DNTK13] in front of members from the Technology Enhanced Learning domain participating at the conference, disseminated within the Metis project among IMS LD authoring community experts and presented at the 9th International Conference on Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom 2013) [NDK13]. The SyncLD approach represents a big first step towards a domain-independent conceptual modeling approach and has been well received by the involved communities. It is part of a pioneering work on providing support for NRT collaborative editing of learning design models. Technically, we used an XMPP-based approach for managing the collaboration operations and distributed OT algorithms for ensuring data integrity and consistency on all participating sites. The objectives of achieving a NRT collaborative IMS LD authoring through introducing synchronous shared editing on learning designs are fulfilled, as shown in the artifact evaluation. This shows – similar to the Community Application Editor's first iteration – that Web-based IMS learning design authoring applications have advantages over the desktop versions, that synchronous NRT collaboration is a useful feature for LD authoring and community experts do favor the usage of such a tool to design units of learning in the future.

Among the lessons learned, the NRT behaviour and the reliability of the OT library used can be drastically improved, especially in P2P settings. The experience gathered with the

OT library shows that in such distributed infrastructures the shared editing does not perform well enough to be able to sustain the prototype usage past a beta stage. Although during the evaluation the replicas between collaborators were identical and the shared representation converged, the standard deviation of the message propagation speed measurements shows rather high delays. Furthermore, technical tests performed in settings with more than three participants show that errors also may occur during collaboration, for example when synchronization messages are not delivered by the underlying network.

The need for awareness was as well an issue mentioned by several modelers during the evaluation. This can be added to the widget space and display hyperlink information on what each collaborator is currently working on. An example is "User X is editing the description of activity 'Read a book'. User Y is currently assigning the tutor role to support activities." Another linked thread are the nudges to users on what actions they could perform next to proceed in collaboratively completing the model. To achieve this, the awareness information in combination with a model of the dependencies and rules in effect during the IMS LD modeling process to provide those nudges in a widget can be used.

The source code of all components of SyncLD are released with free and open source licenses. This is one of the major differences to projects like the Realtime API in Google Drive, which offers the synchronization infrastructure as a black box, thus obtaining control over the functionality, content and eventually the "fate" of tools which are developed using such black box APIs. In contrast, by releasing open frameworks for NRT communication, application developers will have full control over the communication infrastructure and can contribute to making the Web a place where NRT collaboration is offered in all applications. Finally, since SyncLD was developed chronologically shortly before the widgetizing methodology, it was implemented as a widget-based tool, but it only consists of one widget. By that, the general idea behind our reengineering approach, for separating the different user interface functionalities in individual widgets, is not taken into account. This issue is also addressed in our modeling framework abstraction, in the next iteration.

5.2 Domain Independent Collaborative Conceptual Modeling in NRT

The updated requirements and objectives for the second iteration, which proposes an abstraction of the NRT collaboration framework from the IMS LD context towards the collaborative creation of arbitrary models using flexible modeling processes are summarized below:

- *Domain-independent visual language definition**[2] A meta designer can define the metamodel of the abstract syntax of the visual language in a NRT editor (similar to a UML class diagram)

[2]The * sign marks the most important, refined requirements

- *Custom visual representation* The graphical appearance of the objects of the modeling language can be defined either by selection out of predefined shapes, such as a rectangle or circle, or by manually defining the appearance via code
- *View abstraction** Metamodelers can create a viewpoint in a graphical NRT collaboration manner. They can add, remove and delete view-objects and view-relationships based on a metamodel. They are able to change, add and delete attributes of the view-objects and view-relationships. They can also define conditions for attributes. It is possible to edit the visual appearance of each view-object and view-relationship
- *Nudging abstraction** Metamodelers can create a guidance model based on a metamodel and predefined collaboration strategies
- *Multiple view/viewpoint support* Metamodelers and modelers are able to create and remove an arbitrary number of viewpoints respective views
- *Conditional styles and filters* Metamodelers are able to define condition statements on the attributes of a view and connect them with logical operators
- *Metamodel to model editor generation** The meta-designers can invoke the generation of a created metamodel (the visual language with component's appearance definition) into the model editor, compliant with its metamodel. The metamodel-to-model translation can be invoked at any point upon need
- *Viewpoint to view generation** Upon generation, an arbitrary number of views on a model should be created as well, according to the defined metamodel viewpoints
- *Nudges generation** Upon generation, the guidance language specification defined during the metamodeling phase is instantiated to a set of actions and suggestions that will be shown to modelers, based on their selected collaboration strategy
- *View modeling* Modelers are able to apply a defined viewpoint to a base model to generate a view. Modelers can edit attributes, add new elements to the view in a NRT collaboration manner. The available modeling elements are visualized according to the viewpoint definition
- *NRT collaboration* Multiple users can collaborate on the creation of a metamodel or model in NRT, regardless of their physical location. Changes initiated by a user directly become visible to the other participants. Other users cannot restrict a user in editing the diagram and by that, each user can always modify all parts of the model or a defined view. In addition, a user can join or leave the modeling process at any time without affecting the other users
- *Free hand model editing* Modelers can modify nodes and edges by directly interacting with them in the canvas: Nodes can be selected from a palette and placed anywhere in

the canvas with a click on the desired position. Once placed, nodes can be moved and resized via simple drag and drop operations. Edges can be placed by performing a drag and drop operation from the source node to the target node

- *Structured editing* Nodes and edges can also be added via a context menu. The context menu only shows options which are valid with respect to the metamodel. E.g., when creating an edge via the context menu the choices for the edge type and target node type are restricted based on the definition in the metamodel
- *History of operations* Undo and redo functionality for operations and actions
- *Awareness** During modeling users have access to the awareness features already provided by SyncMeta. An activity list shows actions that have recently been performed by all collaborating modelers. Additionally the currently selected node of remote users is highlighted with a user specific color
- *Componentized Web Interface* The modeling application should contain several widgets, according to the widgetizing methodology principles. Among the core functionality that can be translated into clear-cut frontend elements are a canvas for modeling, a palette containing the modeling language elements, awareness information, nudging information, property editor for modeling elements, import/export, collaboration overview

5.3 Artifact Design and Development: The SyncMeta Approach

Conceptual modeling[3] plays an important role in representing domain-specific information during the requirements elicitation and design phases of information systems [NJJ+96]. Conceptual modeling frameworks reduce system complexity by introducing abstraction levels and by offering sublanguages for different static, dynamic, or goal-oriented perspectives, which allow involved stakeholders to focus on artifacts and concepts of their interest while linking these perspective models through formal metamodels. In addition, the concept of **views** allows stakeholders to focus their attention on modeling areas of interest within or across modeling perspectives. Previous research around the turn of the century formally investigated stakeholder viewpoints and the generation of views from metamodels particularly from a requirements engineering perspective [NJJ+96, NJ99]. For example, certain stakeholders may prefer a different view to express their concerns [NKF03, KW07b], as models in complex systems tend to become quite large. Views are used to deal with this complexity in (meta)modeling frameworks. View-based model-driven design aims at creating partial metamodels and models, each reflecting a set of concerns of the modeled

[3]The content of this section is a summary of [NRD+18]

system [NKF03]. Of course, collaborative development using views raises the need for conflict resolution between possibly conflicting viewpoints within or across modeling perspectives. In addition, there is the challenge of generating coherent code satisfying the requirements specified in an agreed set of views, in order to enable model-driven generation in later development stages, as well as in system evolution.

In this chapter, we revisit the opportunities from the modeling literature such as metamodeling, metamodel to model generation, or nudging and recommendations during the model authoring process from a NRT collaboration perspective. We specifically explore the opportunities and challenges of integrating view abstraction and view-based model-driven development with the NRT setting. Therefore, we propose SyncMeta an approach that allows the collaborative creation of metamodels in NRT based on a visual language specification and the generation of collaborative model editors from the defined metamodels. It facilitates a collaborative, graphical definition of views and guidance on the metamodel layer which can then be applied in models. In several stages of evaluation during an iterative development process of our approach, technical criteria such as flexibility, scalability, and NRT performance have been considered. This has been complemented by user studies in which we asked questions such as "should the view-based approach be constructed with a more centralistic model-view linkage, or with direct peer-to-peer (view-to-view) mappings?", "how important is the NRT aspect to development teams?" or "how to assist users during NRT collaboration modeling processes?". Altogether five such case studies with different kinds of modelers were conducted in parallel to the iterative development process of SyncMeta [DNE+15] a Web-based framework which provides a flexible and performant architecture for achieving domain independent collaborative modeling, and is available in the GitHub repository[4].

5.3.1 Near Real-time Collaborative (Meta)Modeling Approach

An illustration of the concepts and roles in the collaborative modeling process is given in Fig. 5.5. On the metamodeling layer metamodelers use the "Metamodel Editor" for collaborative authoring of a metamodel, represented by a Visual Language Specification (VLS). This builds the basis for a model editor for the specified modeling language. A VLS is defined visually using a graph-based visual modeling language (VML). An arbitrary number of model editors can be generated based on the defined VLS. The NRT collaboration takes place at both metamodeling and modeling layers. On the metamodeling layer metamodelers may collaboratively define viewpoints. For the definition of a viewpoint the underlying VML was extended with additional view types. The view types define references to classes of the metamodel. Here, conditions on the attributes of the referenced class can be applied. The appearance and rules for each view type can be redefined in a view. To facilitate the metamodeling process, a Closed-View Generation (CVG) algorithm based on [Run92]

[4] https://github.com/rwth-acis/syncmeta

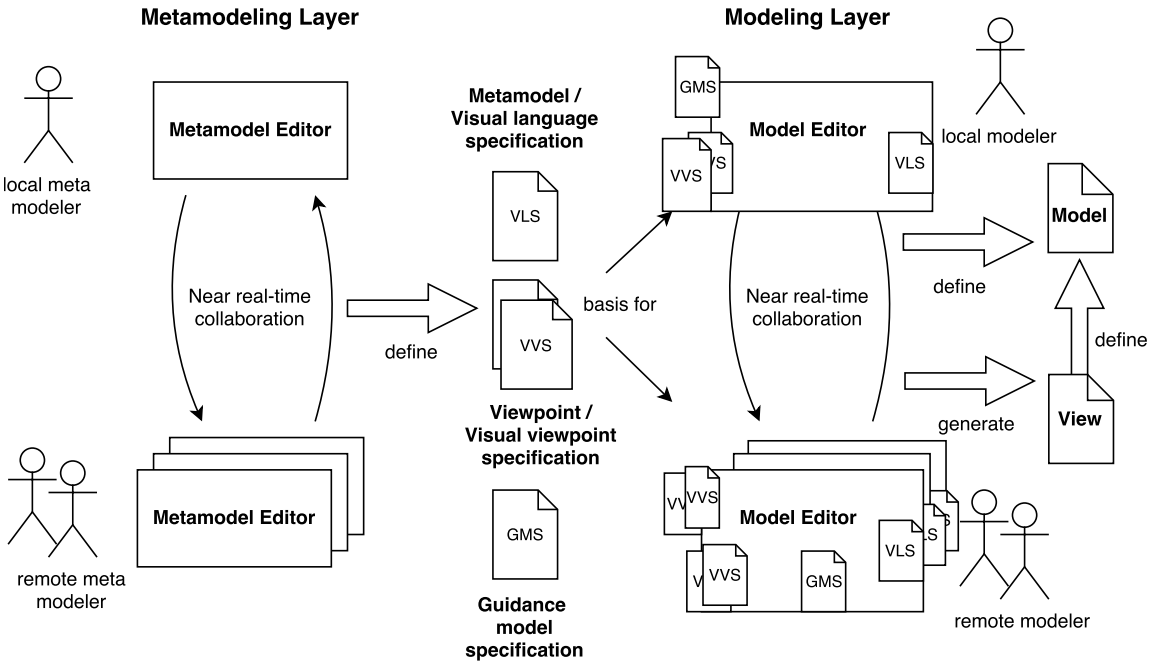


Figure 5.5: Concepts and roles in the SyncMeta approach view-based (meta)modeling process [NRD+16]

automatically adds classes and relationships to the viewpoint when a reference to an object or relationship is defined in the metamodel.

Similar to the VLS generated for a metamodel, a visual view specification (VVS) is generated for each viewpoint, consisting of a construction plan for the view in the modeling layer. Here, an existing model can be used in combination with a certain VVS to generate a view on the model. Modelers may collaboratively edit any view or the model itself in NRT, with all actions being propagated to collaborators and reflected in all views and in the model.

Finally, a Guidance Model Specification (GMS) is defined using a Guidance Modeling Language (GML). An extension of UML Activity Diagrams, the GMS specifies main modeling activities and a flow of (simultaneous) actions that compose such activities. During the model generation, a guidance assistant engine is being instantiated based on the GMS, that can nudge users regarding modeling activities and action to perform. The actual guidance encompasses two types of guidance (i.e., work together/collaborate and avoid conflicts strategy) that are shown to the modelers. The engine decides which guidance is shown to which modeler, based on the performed actions and the actions performed synchronously by other modelers.

5.3.2 Views and Viewpoints

The conceptual modeling literature offers many different uses and definitions of the terms *view* and *viewpoint*, which are often used interchangeably. Based on the existing work, we provide working definitions for these terms and explain the relations between the different concepts used in the visual modeling approach. The definitions are used in Section 5.1.2 for explaining the implementation of the viewpoint modeling and the view generation. In [FPK+12] a *viewpoint* is defined as a language that reflects various aspects of a metamodel. It can restrict the original metamodel and it addresses a set of concerns of one or more stakeholders. The modeling language defined by a viewpoint describes the concerns of various stakeholders and corresponds to a submodel of the metamodel. A *view* is the presentation of a model by applying a specific viewpoint. Thus, a view is a concrete instance of a viewpoint. A *view type* is a meta class whose instances a view can display [GBB12]. More precisely a node or edge of a view constitutes an instance of the view type defined in the corresponding viewpoint. Thus, a view type is a derivation of an object or a relationship class that comprises a set of rules. These rules can be selectional or projectional predicates that determine the representation of an object within the view. Furthermore a viewpoint is defined by a collection of view types.

These definitions are slightly more narrow than the IEEE 1471 Standard [ISO], as views only visualize and update a single central model under study. In contrast to IEEE 1471 standard, a view will not consist of more than one model. In general, we conform to the IEEE Standard such that a viewpoint constitutes a set of rules and conventions for the specification of a view. Furthermore, a view is a representation of the model under the perspective of a viewpoint.

In the following, we introduce the formal definitions for the terms introduced above. First we define the sets of classes, properties and types. Finally, we define the formal concept of a metamodel.

Let P be a set of all possible properties. Each $p \in P$ can be an arbitrary complex function or a simple value from an enumeration type. We only require that each p has a label, a type and a unique identifier. Let T be the set of all types defined in the VML on the metamodeling layer of SyncMeta, e.g. $T = \{Object, Relationship, NodeShape, Generalization, Association, \dots\}$. An overview of all types in the VML is depicted in Fig. 5.6. Let C be a set of all possible classes. Any class $c \in C$ has a unique name, a type description, and a set of properties. We define the signature of a class c as a triplet with $c = (l, t, A)$, where l is the unique name of the class, $t \in T$ is the name of a type associated with the class, and $A \subset P$ a finite set of properties. We also require two simple access function for the label and the type of a class $c \in C$. We define $label(c) = l$ for $c \in C$. Furthermore, we define $label(C) = \{label(c) \mid c \in C\}$ as the set of all unique identifiers of all classes in C. Analogously, we define $type(c) = t$ as the type of class c and $type(C) = \{type(c) \mid c \in C\}$. These access functions are mainly used to describe the relation between a metaclass of the metamodel and their instances in the model. The properties of a class do not play any role in

the constitution of the relation between a metaclass and the metamodel, therefore an access function is not required. Similar definitions can be found in [Run92].

Definition 5.3.1. *A **metamodel** is a directed graph $G = (V, E)$ with $V \subset C$ a finite set of nodes and $\forall c \in V : type(c) \in T$. E is a finite set of edges with $E = \{(l, t, c_1, c_2, A) \mid c_1, c_2 \in V \wedge t \in T \wedge l$ an identifier $\wedge A \subset P\}$.*

We assume that a metamodel may consist of an arbitrary number of classes and each class may consist of an arbitrary number of properties. We only require that the type of each class belongs to the VML. We define a viewpoint in a similar manner. A viewpoint is a metamodel on its own. We also require that a viewpoint consists of at least one view type. Thus, we formally define a view type before we give a formal definition of a viewpoint. We define a function φ that transforms an object class or a relationship class into a ViewObject or ViewRelationship class, respectively. On other classes the function φ is the identity function.

Definition 5.3.2. *Let $VT_C = \{\varphi(c) \mid c \in C\}$ and $\varphi(l, t, A) = (l', t', A', l)$ with l' the new unique label, t' is ViewObject or ViewRelationship if t is Object or Relationship, else $t = t'$. Obviously, $A' \subseteq A \subset P$.*

A view type class of a viewpoint consists of a reference to a class in the metamodel. The reference is the unique name l. Thus, a viewpoint is not independent of the metamodel.

Definition 5.3.3. *A **viewpoint** with respect to VT_C is a metamodel with $G' = (V', E')$, and $\exists c \in V' : c \in VT_C \wedge type(c) =$ ViewObject.*

Based on these formal definitions of the concepts at the metamodeling layer, we can define the concept of viewpoint applied to a model of the modeling layer. For the generation of the model editor instance a VLS of the metamodel is generated. For simplicity, we consider the VLS to be the metamodel described in Definition 5.3.1. First, we formally define the relation between the metamodel defined in the metamodeling layer and the model.

Definition 5.3.4. *Based on graph $G = (V, E)$ of a metamodel, a **model** is a directed graph $M = (V', E')$ with $\forall v \in V' : type(v) \in label(V)$ and $\forall e \in E' : type(e) \in label(E)$.*

We require each node and each edge of the model to be an instance of a node type or edge type defined in the metamodel. To generate views we first need to define a function that applies a view type to an entity within the model:

Definition 5.3.5. *Let $VP = (V, E)$ be a VVS of a viewpoint. Let $\phi_v(n)\colon (l, t, A) \mapsto (l, type(v), A')$, $v \in VT_V$ is a view type class of VP. $A' \subseteq A$ is the new set of properties defined by view type v. Analogously, the function for edges is defined as $\phi_v(e)\colon (l, t, c_1, c_2, A) \mapsto (l, type(v), c_1, c_2, A')$, where A' is generated from the attributes defined for v.*

With this helper function we can define a view as follows:

Definition 5.3.6. *Let $VP = (V_{VP}, E_{VP})$ be a VVS of a viewpoint and $M = (V_M, E_M)$ a model. A* ***view*** *$V = (V_V, E_V)$ is a subgraph of M with $V_V = \{\varphi_v(c) \mid v \in V_{VP} \wedge c \in V_M\} \subseteq V_M$ and $E_V = \{\varphi_v(e) \mid v \in V_{VP} \wedge\ e \in E_M\} \subseteq E_M$.*

The resulting view is a subgraph of the model it is applied on. Each node/edge whose type is referenced to a view type in the VVS is part of the view. For the view generation, we need a VVS and a existing model as input. In contrast to other approaches like ArchiMate [Lan13](cf. Chapter 2), every manipulation of an element of a view is directly reflected into the model. This holds for both visual manipulations (for example resizing and dragging of nodes) and data manipulations (for example changing values of attributes or creation of nodes and edges). Furthermore, our view-based modeling approach follows a simple one-to-one mapping between elements of a model and a view. A concrete view type of a viewpoint is directly mapped to exactly one Object/Relationship class of the metamodel. This means that in the case of the modeling layer an element of a view is directly mapped to one element of the model.

5.3.3 Metamodeling and Viewpoint Definition

In the previous sections we have presented our main SyncMeta approach and the formal definitions of viewpoints, views and view types. Extending our earlier work in [DNE+15], we offer detailed information on the used metamodeling hierarchy, which adopts a similar approach to frameworks such as ADOxx [FK13], AToM [LV02] and the one specified by the MOF [AK03]. The core idea is abstraction, the superior level defining the constructs and rules that are to be applied in the lower levels of the hierarchy. Thus, the metamodel defines the model elements available in the model editor and the metametamodel or any other more abstract metamodels thereof specify the rules and elements implemented in the graph-based modeling approach. To describe the graph-based modeling language that a model editor should be generated for, a particular visual model description language (VMDL) is used. This modeling language is based on UML class diagrams [RJB04] to reduce the effort required by novice users in learning a new modeling language. However, to increase the usability and flexibility of our visual-based modeling approach we have introduced certain entity types. An example for this is the “Abstract Object”, which in our case behaves like a container for introducing generalization within the VMDL. The object makes it easier for modelers to keep the overview of the model taxonomy, i.e. specific types of objects. The used VMDL is closely related to the representation of its models as graphs. Thus, it supposes a one-to-one relation between the elements of a model, i.e. objects and relationships, and the elements of a graph, i.e. nodes and edges. There are other approaches like the one presented in [Min07], that allows the implicit definition of concepts of the model, for example by defining a relationship by the objects it links. However, this requires a deeper analysis and correctness checking of the defined model and goes beyond the NRT collaboration for view-based conceptual modeling focus of this paper. We implement a four-tier metamodel hierarchy, which is depicted in Fig. 5.6.

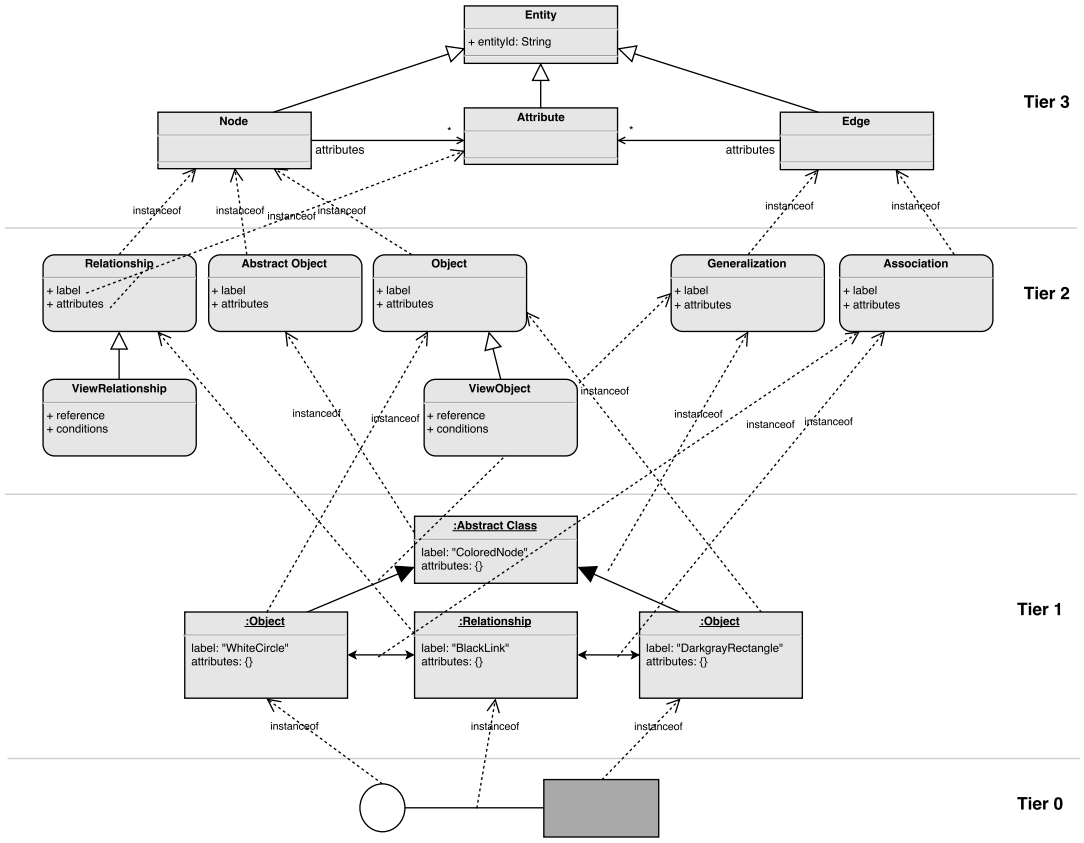


Figure 5.6: Simplified extended metamodel hierarchy with view types

Tier 3 constitutes the topmost tier and defines the basic elements of the graph-based VML. A modeling language consists of nodes and edges, defined here as entities (i.e. Node and Edge entities). As the framework enables the definition of node and edge attributes, a third entity Attribute is part of this tier. To each Node and each Edge can be assigned an arbitrary number of attributes. The entity Attribute just constitutes an abstract representation of the attributes actually used in the framework.

Tier 2 is the metametamodel and defines the node and edge types of the VML as well as the view types of the viewpoint models. It contains the elements of the visual modeling language used by the metamodel editor to define the modeling language an editor should be generated for. As stated in Definition 5.3.2, a viewpoint does not contain any *Object* or *Relationship* types. We replace them by using the *ViewObject* and *ViewRelationship* types, which are a specialization of *Object* and *Relationship*, respectively. These contain a reference to a node type or an edge type in the metamodel. It is also possible to define conditions on the attributes of the referenced class, i.e. in contrast to a simple object class a view-object offers functionalities to customize the attributes of a view. Metamodelers are thus able to hide and rename attributes. The *Conjunction* attribute determines the logical connector of the

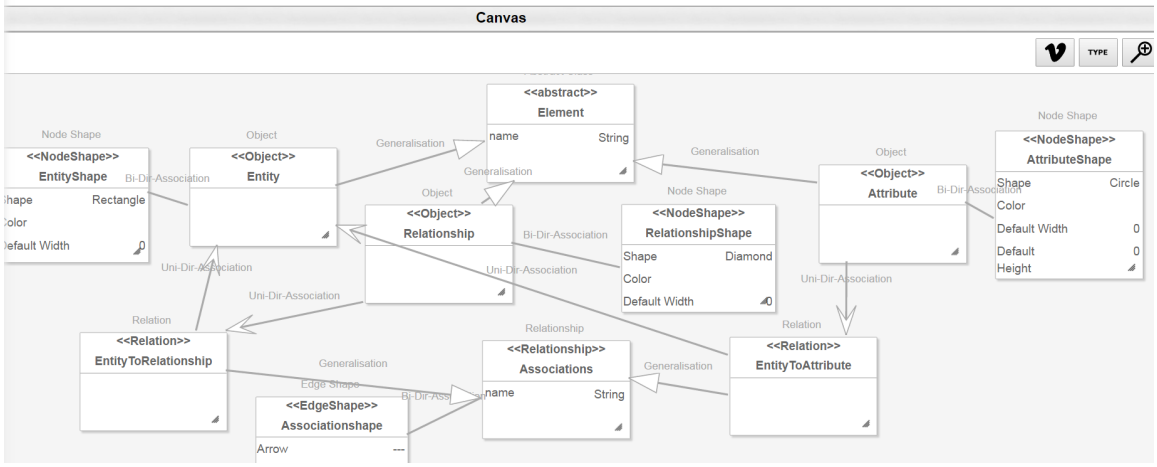
conditions. This can be either the logical `AND` or `OR`. Thus, we can build a formula with the predicates $\varphi_1, .., \varphi_n$, either with a conjunction over all predicates $\varphi_1 \wedge ... \wedge \varphi_n$ or with a disjunction over all predicates $\varphi_1 \vee ... \vee \varphi_n$. Conditions on attributes allow metamodelers to make simple queries on the attributes of an object class and filter the entities of this class in the view canvas of the model editor. With auxiliary classes it is possible to define custom node and edge shapes for each view type. Each concept of Tier 2 can be simply converted to an UML class diagram (assuming that each *Relationship* is associated with one ingoing and one outgoing *Object*). We can map *Object*, *Relationship* and *Abstract Object* to an UML class. The *Abstract Object* has the property abstract in the corresponding UML class. The *Generalization* relation is mapped to a class diagram association. For each *Relationship* convert the ingoing *Association* to an outgoing UML class diagram associations labeled with "source". Outgoing *Associations* are simply mapped to UML class diagram associations labeled with "target".

Tier 1 defines the actual metamodel or viewpoint. The elements of this tier are instances of the elements of Tier 2 and will be instantiated by the elements of Tier 0. In the figure, a simple modeling language is defined on Tier 1 consisting of two Objects and one Relationship. The Relationship is associated with both Objects. By that, theses Objects can be linked to each other on the modeling layer (Tier 0). The two Objects are generalized by an Abstract Object node, whose attributes they inherit. The Objects and Relationships have to be linked to instances of Node Shape and Edge Shape to define their visual appearance. Therefore, for this example three additional nodes, that are not shown in the figure, lie on Tier 1, ensuring that the Objects labeled WhiteCircle and DarkgrayRectangle are drawn as a white circle and darkgray rectangle, respectively, and that the Relationship labeled BlackLink is represented by a black link. Metamodelers are allowed to develop an arbitrary number of viewpoints in the same NRT collaborative fashion they are used to define metamodels. The metamodel is the input for the model editor instantiation and each viewpoint is generated to a VVS.

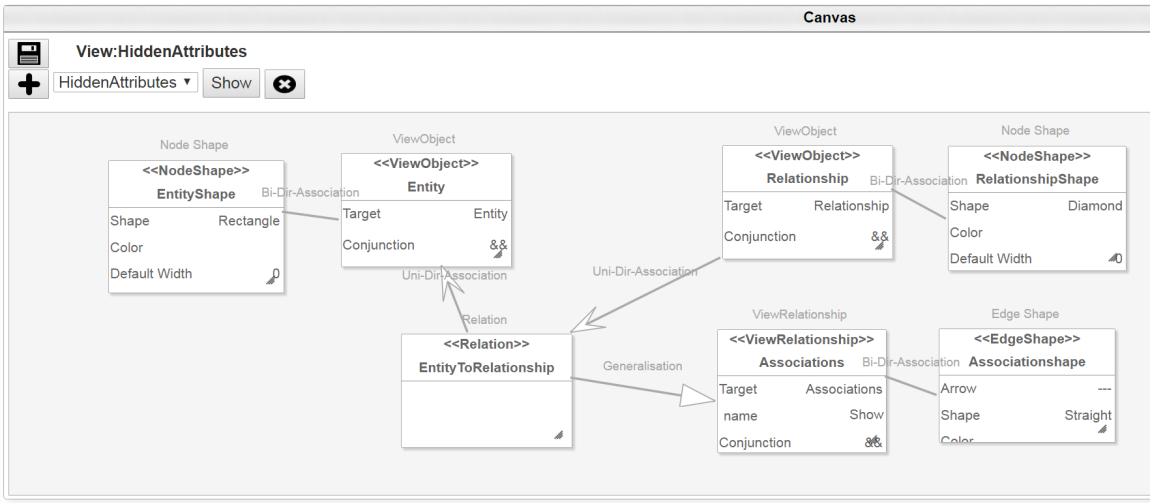
Fig. 7.6 depicts a more detailed screenshot of a metamodel of the Entity Relationship (ER) diagram. It also illustrates the concept of a Relation-node. A Relation allows more fine-granular relationships between an Object and Relationships. If we connect all associations with the Relationship-node, we will allow connecting Attributes with Relationship. The Relation concept allows more detailed source-target-relations by generalizing the desired Relationship.

Fig. 5.7(b) depicts a very simple viewpoint for the ER diagram. The idea is that all Attribute nodes of an ER model should be hidden. Therefore metamodelers simply add ViewObjects and ViewRelationships to the viewpoint of the object and relationship types which should be visible in the particular viewpoint. The Closed-View Generation algorithm automatically adds auxiliary classes (such as Node and Edge Shape and Relation) and their associations to the viewpoint by specifying the Target attribute of ViewObject resp. ViewRelationship. By default the auxiliary classes are simple copies of the classes in the metamodel, but they can be further customized in a viewpoint.

Tier 0 is the actual model of the modeling layer, created using a generated modeling editor.



(a) Metamodel for the Entity-Relationship (ER) Diagram



(b) A simple viewpoint that hides attributes

Figure 5.7: Screenshot of ER metamodel and viewpoint definition

In Fig. 5.6, the model simply consists of two nodes and one edge linking both nodes. One node is an instance of the Object labeled WhiteCircle on Tier 1, the other node is an instance of the Object labeled DarkgrayRectangle on Tier 1. The nodes' appearances correspond to the shape definition associated to the elements that they are instantiated from. The same holds for the edge, which is an instance of the Relationship labeled BlackLink on Tier 1. On Tier 0 a viewpoint is applied to the model. The resulting view supports NRT collaborative modeling as well. While concepts on Tiers 2 and 3 are implemented in the framework, models on Tier 0 and 1 are defined by modelers and metamodelers, respectively.

5.3.4 Guidance Modeling

We constructed a guidance approach allowing metamodelers to define modeling guidance (GML) and generating an intelligent assistant, that helps modelers during the collaborative modeling process based on the guidance definition. We use a form of elastic collaboration [JLB12], which is a combination between workflow-based collaboration (in our case defined by metamodelers) and emergent collaboration, where collaborators have to coordinate themselves in order to perform certain actions. The elastic collaboration assumes that collaborators can use the two basic collaboration types intermittently, depending on their needs and that the system monitors the collaborative behaviour in order to detect transitions. The guidance model is similar to the NATURE process metamodel [PR95] and the PRIME framework [PWD+99] for defining process-integrated modeling. The assistant monitors the actions of modelers (such as selected element, currently visible canvas elements, mouse cursor position), determines the actions which correspond to the available definition and computes the guidance elements that can be shown accordingly. The guidance is presented in the form of nudges [TS08]. The provided guidance helps modelers to perform actions faster, get suggestions on which steps they can perform next during modeling, while keeping into account the collaborative setting. The guidance model is defined by the metamodelers in the metamodeling step, based on the visual language specification. This is done using a NRT collaborative guidance model editor that offers the same features as the metamodel editor. The GML is based on UML activity diagrams [RJB04, Obj15], such that modeling actions and threads can be specified as activities. It can be regarded as a SyncMeta specific instance of an UML activity diagram. An activity is represented by a sequence of non-linear actions. The actions model the available SyncMeta object manipulations (such as actions performed on objects, properties or relationships). The various flows between actions and activities are augmented using control nodes. For this purpose we use decision nodes to define branches in the modeling flow of possible actions. The fork node is used to define collaborative, concurrent actions. The action nodes and the control nodes are connected to each other using control flow edges. These determine the ordered sequence of actions. The GML also contains object nodes, that can specify a flow of objects, or entities (the term entity is used to differentiate between the SyncMeta metamodel's elements). The "EntityFlowEdge" is used to model the flow of objects/entities. The following UML metamodel elements are used (cf. Fig. 5.8, Fig. 5.9):

- Activity nodes (as an instantiations of the activity UML metamodel element). The activities nodes can be specialized to "Create Entity Node", "Call Activity Node" and "Set Property Node". The "Create Entity Node" is used for creating SyncMeta "Object" and "Relationship" nodes. The "Set Property Node" nodes are used for the properties of SyncMeta objects
- Control nodes, as a type of Activity node, that are specialized –according to the UML specification– into "Initial Node", "Activity Final Node", "Decision Node" and "Fork Node". These nodes are used to manage the flow of activities during the

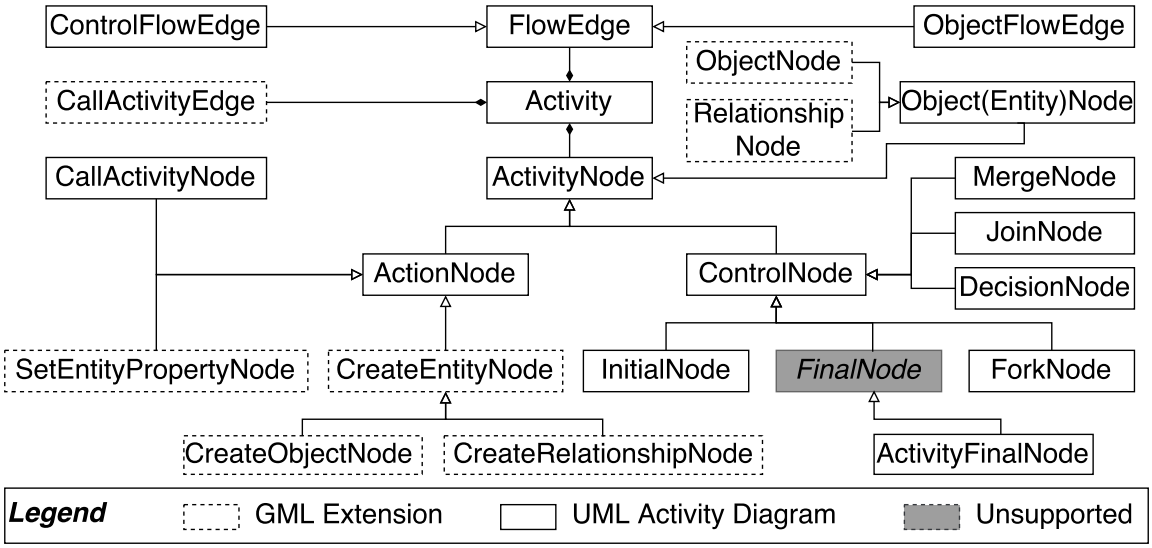


Figure 5.8: Activity nodes used in the guidance metamodel according to the UML activity diagram metamodel

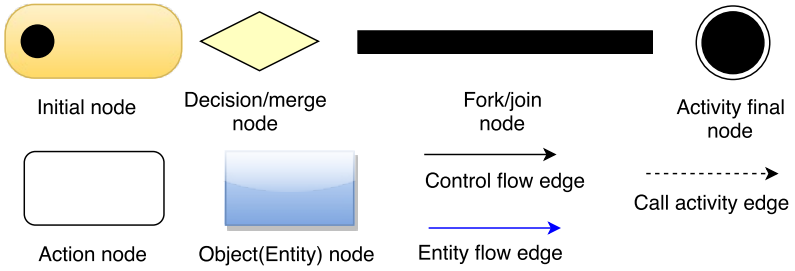


Figure 5.9: Guidance modeling language notation

collaboration and –using the “Call Activity Node” to switch between the various possible collaboration flows

- Flow edge represents the flow of control, i.e., transitions between actions part of an activity, represented by the “ControlFlowEdge” and the “ObjectFlowEdge” that represents the flow of entities or SyncMeta objects
- “CallActivityEdge” is added to represent and make it easier for users to link elements such as “CallActivityNode” with an “InitialNode” of another activity

The GML therefore needs to be defined by metamodelers for each language.

The semantics of objects and their relationships need to be considered when designing the guidance, as well as the flow of actions, possible activities and their dependencies. The specific actions depend on the objects, relationships and properties expressed using SyncMeta’s metamodel editor.

Compared to UML activity diagrams, the GML is more restricted, as it does not support features such as object flow between actions, multiple initial nodes for an activity, preconditions, etc. Further GML limitations include the lack of object flow between activities and the support restricted to only creation or edit of objects/entities (i.e., delete actions are not supported). Moreover, repositioning or node resize actions are not supported and cannot be expressed in GML.

5.3.5 Model Editor and View Generation

The model editor comprises all the concepts of **Tier 0** described in Section 5.3.3. In contrast to the metamodeling layer the model editor comprises generic components that are initialized with a specific VLS. Moreover, on this layer modelers may apply the viewpoints defined on the metamodeling layer. For any existing model, this is done by enabling the views in the canvas and selecting the desired view from a drop-down menu (see Fig. 6.5(b) and 6.5(c)). As described in Definitions 5.3.5 and 5.3.6, part of the view are all nodes and edges of the model that are associated with a view type in the corresponding viewpoint. In this respective view, all other nodes and edges are hidden. Apart of filtering on the type level, we allow filtering nodes and edges on the instance level based on their properties values. Accordingly, custom styles such as adjusting the color, shape, labels or connectors can be applied on the selected view. The following steps are used:

- Filter nodes/edges regarding the ViewObjects/ViewRelationships of the VVS
- Filter nodes/edges by conditions defined on their attributes
- Apply custom styles for each node/edge

A screenshot of the view-based editor is shown in Fig. 6.5, described in the Community Application Editor realization. A further view generation example of an i^* model that was used in the evaluation of the view-based modeling approach is depicted in Fig. 5.16.

5.3.6 Guidance Assistant

During the model and view generation, the modeling guidance is generated based on the GML defined in the metamodeling phase. According to the metamodelers specification (which can typically be regarded as domain experts that have the domain knowledge to be able to define relevant guidance activities), the guidance helps during modeling by increasing the speed during the modeling process through customized, 1-click suggestions presented as an additional toolbar in the modeling canvas, positioned next to a selected node or edge. It can also suggest common modeling activities (for example to novices that do not have expertise with a certain modeling language, such it is the case for IMS LD models) by

Operation	Purpose	Conflicting
EntitySelect	selection of a node or edge	
Node-/EdgeAdd	creates a node/edge	
Node-/EdgeDelete	delete a node/edge	✓
NodeMove(Z) & NodeResize	moves/resize a node	✓
AttributeChanges	change the value of attribute	✓

Table 5.4: Operations occurring during (meta)modeling in SyncMeta

suggesting activities and modeling steps that can be performed. The nudging mechanism is achieved by only offering suggestions to modelers, which they may decide to follow or not. We do not enforce actions or activities, nor do we restrict the modeling features in any way. The assistant supports four guidance items: *object guidance* (suggests a modeling object that a modeler can select), *property guidance* (suggests modelers to add properties to objects or edges), *edge guidance* (suggests to add certain relationships between objects) and *collaboration guidance* (suggests modelers to support other modelers involved in a certain activity). The collaboration guidance performs suggestions according to certain predefined strategies. The collaboration strategy suggests actions that try to nudge modelers to work together in the same modeling activity. As an example, the assistant tries to help one user to define modeling objects in the canvas, another one to create relationships between the objects and a third user to define properties of nodes and edges. The second strategy tries to avoid conflicts by nudging users to work in their own modeling context/activity and do not interact with other modelers.

5.3.7 Near Real-time Collaboration and Conflict Resolution

Our approach enables non-locking collaboration — each user can manipulate any part of the model at any time. It follows an optimistic perspective on collaboration, as agility and flexibility are preferred over more complicated rules and slower processes (such as locking parts or entire model, turn taking, versioning/merging, semantic checks, etc.). As in the case of SyncLD, the collaboration is enabled for each model, in a dedicated "collaboration space", where modelers (i.e., community members) can join at any time. Each modeler has its own model copy stored locally, where all changes (local and remote) are being applied. Operations are propagated live using standard Web protocols between all stakeholders involved in authoring a model (with or without views). Therefore, they are visible in NRT by all participants, regardless if they are working on the model or within a view. We categorize all possible modeling actions into conflicting and non-conflicting operations. A list of all operations which occur during the (meta)modeling, viewpoint modeling and the view definition processes is depicted in Table 5.4.

Actions performed on a model that can be directly applied without any further computations or changes are non-conflicting. An example of such an action is the node creation, as the

nodes are uniquely identifiable within one model and their creation is intentional. During the (meta)modeling process, a number of operations can lead to high level conflicts, for example when two or more collaborators edit the same text attribute at the same position. Such conflicts are automatically resolved in NRT by the framework implementation, guaranteeing a convergent document state. As it can be observed, another conflicting example scenario could be that two or more collaborators move the same node at the same time. Considering that another user concurrently deletes the same node, the occurring conflict needs to be resolved and each modeler needs to end up with the same representation of the visual modeling graph. In this concrete example, the delete operation removes the shared data of the node in the shared document state. Concurrently, the move operations alter the primitive data of the shared values of the node and will not be considered anymore. During the implementation, the importance of certain actions (deletions, node changes, etc.) can be influenced. An example is the deletion of nodes, which we considered to have a higher priority during collaboration as move or edit actions. If in a view or model a node is deleted during the collaboration, the deletion is performed even if another user may set an attribute on the respective element. However, all modelers can undo the applied operations, with the model remaining consistent (for example existing operations are applied in inverse order). Optionally, when performing a deletion, a violation check on the model can ensure that an element marked for this operation has not any invisible dependencies in other views and if this is the case, forbid the action. Finally, if a modeler joins the collaboration at a later point in time, he will directly see the existing model or view, together with the history of performed operations.

As mentioned in SyncMeta's introduction, the concept was realized during several iterations, each one trying to solve certain challenges determined after testing and user evaluation sessions (cf. Section 5.5). First iterations [DNE+15, NRD+16] use the same OT engine and a similar implementation to SyncLD for solving conflicts during the NRT collaboration. For each collaborator, the algorithm transforms remote operations before executing them on the local document copy and ensure that all local and remote copies are identical. However, as shown in the evaluation of our first iterations, such algorithms proved less efficient than conflict-free replicated data types in P2P settings, where our approach can also be used. OT saves every received operation in a state vector which is then always transmitted with every operation. In P2P environments the vector state grows proportionally with the number of collaborators. Moreover, a global total ordering of the operations is obtained using timestamps and vector clocks, which sometimes can lead to multiple steps required to achieve consistency. Due to the implementation logic needed to transform the graph representation of models to a linear structure that can be synchronized using OT, the framework' collaboration features required considerable implementation efforts. Due the promising alternative solution offered by CRDTs in distributed environments, we have re-engineered our approach using Yjs [NJDK16] in SyncMeta's concept and implementation. The algorithm operates on the data type level representation of the shared model, using lists and map data types to build the graph representation, as presented in the next section. Therefore, it is much easier to ensure conflict avoidance using this model representation.

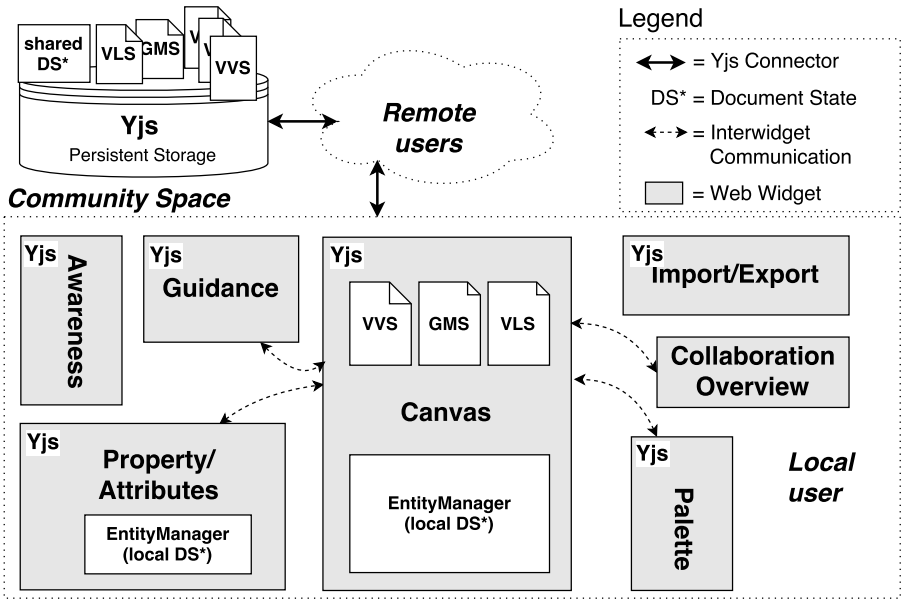


Figure 5.10: SyncMeta architecture: user interface widgets and communication infrastructure

Furthermore, the approach benefits from the advantages of our Yjs implementation over OT solutions, presented in Chapter 4

5.4 SyncMeta Artifact Realization

The SyncMeta implementation architecture targets easy portability, flexibility and reusability. As such, it adopts the proposed widget-based implementation, running on modern Web browsers. However, this can be extended using various Web-based frameworks. Fig. 5.10 depicts an architectural overview of the user interface widget components (depicted in gray boxes) offered by SyncMeta. It also represents how the persistence and communication layers are realized within the infrastructure. As the setup is very easy and their implementation is highly reliable, we chose to use WebSockets as a communication protocol to propagate messages to collaborators, based on the Yjs WebSockets connector. The server supports persistent storage with levelDB and the Yjs connectors fall back to Hypertext Transfer Protocol (HTTP) communication if WebSockets are not supported. Obviously the major disadvantage is that the central server is the single point of failure, but upon need this can be replaced with peer-to-peer messaging solutions.

Each widget connects to a Yjs WebSockets server with the Yjs connector and joins a shared room. The shared room is actually conceptually a one-to-one representation of a community space, that also corresponds to one model authored in SyncMeta. Once the connection is successfully established the widget grants access to all the shared data. SyncMeta uses

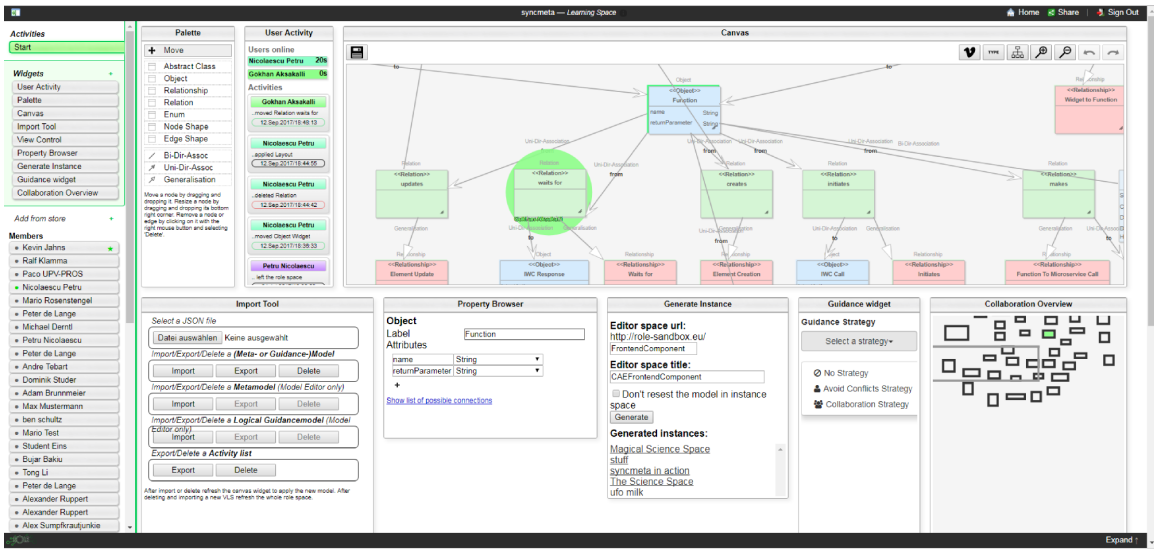


Figure 5.11: SyncMeta metamodel editor screenshot for the Community Application Editor metamodel

shared Map, Array, and Text data types provided by the Yjs framework. These simple types allow the composition and synchronization of any complex data structure, like graphs, text, SVG, required by SyncMeta. The core shared data structure of SyncMeta consists of a shared map which contains the VLS, GMS and VVS stored as a JSON object. These are required to initialize widgets and apply viewpoints to a model (cf. Section 5.3.5).

The most important element is the shared document state, which reflects live the model and actions performed on it. Each entity (node or edge) of the model is represented as a shared map in the shared document state of SyncMeta. A key-value pair of a shared map either consists of a primitive data type (number, boolean, strings) or a shared data type (shared map, array, text). For example a shared map of a node consists of primitive data like height, width, position on the canvas and type of the node. Attributes of a node are represented by shared text object as values and the attribute names as keys. Analogously, each edge of the model is represented as such a shared map containing edge specific primitive data like type, source and target node, and attributes as shared text. The visualization of nodes of edges can be customized in the metamodel editor using SVG code, that can be also edited collaboratively. This feature is implemented using a Yjs Y-text element.

A screenshot of SyncMeta metamodel editor, showcasing the Community Application Editor frontend component metamodel, exposing all the core SyncMeta widget frontend elements is shown in Fig. 5.11. The canvas widget visualizes the current shared document state of the model and provides operations to manipulate the model—e.g. adding nodes and edges, drag & drop, and all other core modeling features. Each edit that alters the model is propagated to other widgets and to other collaborators using Yjs or IWC messages. The canvas widget manages the local document state of the model with the help of an

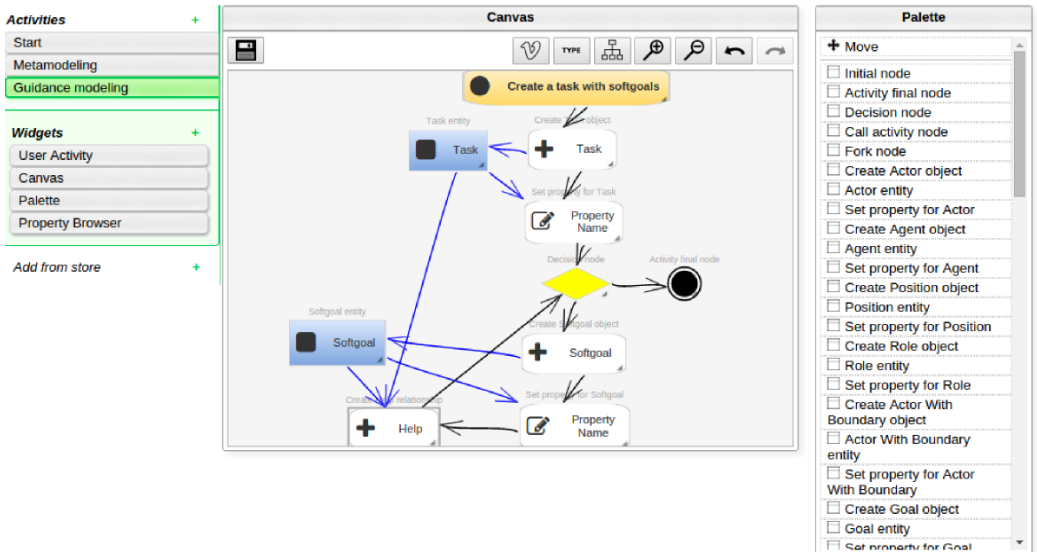


Figure 5.12: SyncMeta guidance modeling screenshot for ER Diagrams

EntityManager. The local document state is always initialized using the shared document state and every event which changes the shared document state changes the local state as well. The EntityManager provides utility functions to traverse and analyze the current (meta)model, which are required by features such as the closed-view generation or the SyncMeta guidance generation. An example of a guidance model specification is presented in Fig. 5.12. To make modelers more aware of recent modifications by other modelers, we have implemented visible traces of user activity in the canvas. Each modeler has assigned a distinctive color once he/she joins a modeling space/session. The color is used as a background for the current objects or edges a user is currently working with. The same color is also used in the activity list widget, which presents information of online users currently changing the model and a list with atomic actions performed by each modeler, using the assigned colors. The traces add a colored circle in the background, which is intensive once the modelers' activity starts and that fades over time. This helps other modelers working on the same model to recognize the activity of other collaborators, without having to focus on the activity list widget, where a more complete awareness information is being presented.

The property editor widget allows editing properties of node and edges selected in the canvas widget. Each property modification (such as changing the title of a node) is propagated back to the local canvas widget as well as to the property editor and canvas widgets of all collaborators. Analogously to the canvas widget, the property editor manages a local document state with the help of the EntityManager.

The palette widget provides the nodes and edge types defined in the metamodel. It dynamically adjusts to the types defined in a particular VVS whenever a viewpoint is applied to a model. Some operations among the canvas, property editor and palette widgets are not

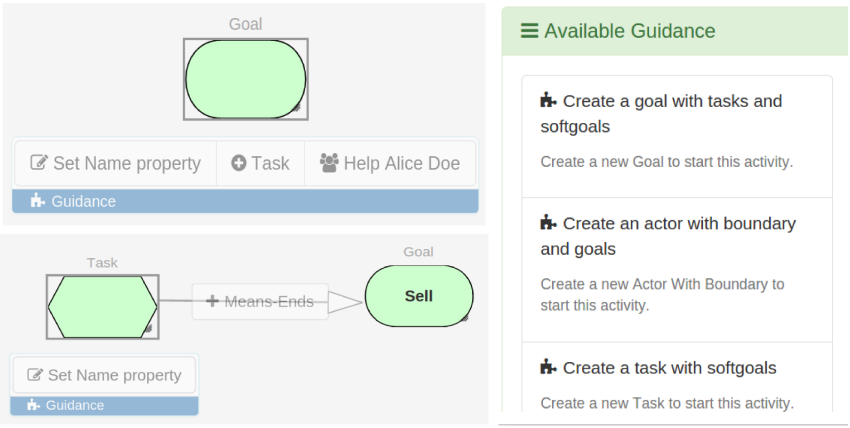


Figure 5.13: SyncMeta guidance modeling screenshot for ER Diagrams

required to be propagated to remote collaborators, for example when the local user selects a node or edge type of the palette. These kinds of operations are only locally propagated using local IWC.

The activity widget tracks and displays the edits made by all collaborators. This is mainly for awareness purposes. SyncMeta consists of several additional widgets which serve special purposes, for example the export widget allows to export a model in JSON or PNG format. The view control widget allows to generate, export, and import a viewpoint metamodel or a VVS.

SyncMeta also includes for awareness purpose a user activity overview widget, that offers a "map" of the entire model and highlights the model parts which are being edited by remote users. The widget is linked to the model representation and offers a preview that can be minimized or maximized.

Finally, the guidance widget that offers action nudges according to the selected strategy and examples of nudges presented in the modeling canvas are presented in Fig. 5.13.

5.5 Evaluation

We have performed the evaluation of the SyncMeta approach during several design iterations, both technical and empirical, conducted with various groups (i.e. student developers and researches within our department, external researchers, project partners, entrepreneurs and persons without technical knowledge). From a technical perspective, the NRT shared editing algorithms adopted in our realization were already evaluated in other contexts. The OT library used in the initial implementation of SyncMeta was also used and evaluated using the SyncLD artifact. The advantages and disadvantages of this library are presented therefore in Section 5.1. The YATA algorithm and the Yjs implementation (cf. [NJDK16]) are presented

in the previous chapter, as they build the NRT collaboration support developed for SyncMeta and the Community Application Editor approaches. Yjs has been evaluated for its reliability and scalability proved to output a good time complexity compared to similar CRDT and OT approaches. It can be used by up to hundreds of users working simultaneously and it can handle both a large number of operations as well as single operations with large content in NRT. Such features are needed in our framework for actions such as pasting large texts under node properties or when the visual representation of nodes (large SVG content) is set during modeling.

We have therefore concentrate our efforts in studying the framework behavior using user studies, in order to validate the technical criteria and determine its feasibility and usability in simulated or real-world usage with real modelers. The user studies aimed to answer questions such as: "is collaborative modeling useful?"; "is view-based collaborative modeling more efficient than modeling without added views"; "is view-based modeling better accepted in collaborative scenarios"; "should the view-based approach be constructed with a more centralistic model-view linkages, or with direct peer-to-peer (view-to-view) mappings?", or "how important is the NRT aspect to development teams in agile scenarios?".

All in all, we have conducted two preliminary studies without views to assess the user interface and the NRT collaboration features of SyncMeta and one study which compared the collaborative modeling with and without enabled views. A further study for assessing the awareness and nudging functionality was also conducted on SyncMeta without the views enabled. Finally, the studies performed using our CAE model-based Web engineering framework for agile development processes based on our SyncMeta is presented in Chapter 6.

5.5.1 Preliminary Evaluation of Basic Features and Usability

During the first evaluation, conducted using SyncMeta without views, 20 student developers worked in groups of two to collaboratively produce an i^* model on seller/buyer relationships in markets. The modelers were located in the same room, each working on a portable computer. They could not observe each others screens, but they were allowed to talk to each other. The results, summarized in Fig. 5.17 were captured using a survey containing 20 questions on the usefulness and usability of the editor. The ratings were based on a seven-point Likert scale with options ranging from very bad (1) to very good (7). An analysis of the produced models showed that the resulting models created by each participant were equivalent. Overall, the editor received a favorable rating ($M = 5.69; SD = 1.17$). Due to the high ratings achieved for the user interface ($M = 5.85; SD = .99$), the usability ($M = 5.7; SD = 1.17$) of the editor and the awareness features available in the framework, these are still present in the latest view-enabled implementation of SyncMeta. However, the simplicity of recovering from errors the user made received the lowest average rating of all questions at $M = 4.95$ ($SD = 1.57$). This was also a major incentive in switching to the Yjs framework for managing the model synchronization. Finally, improving the conceptual

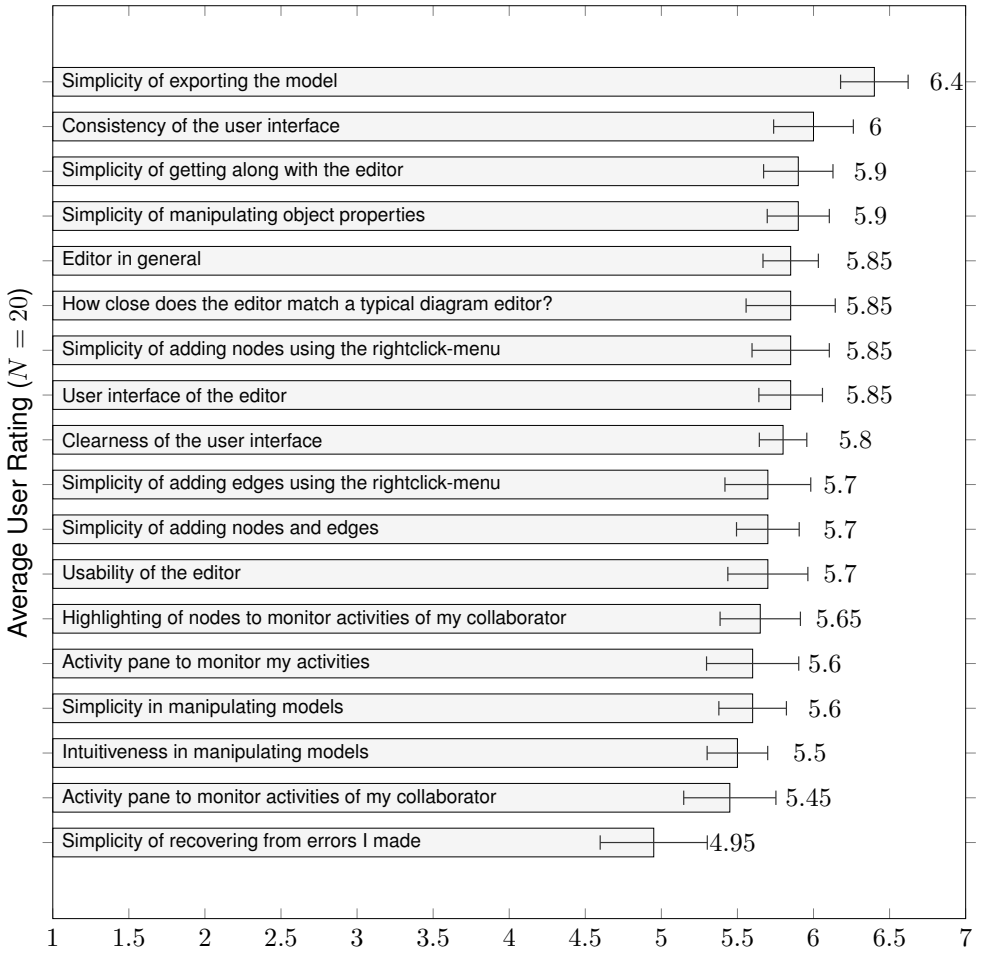


Figure 5.14: Survey results of i^* group sessions (quantitative items) [DNE+15].

modeling capabilities by introducing views in the modeling canvas was among the future work resulting from this study.

In order to study the user interface of SyncMeta and its domain-independent visual modeling proposed approach, the framework was evaluated in the context of the LLP European Project "Metis"(531262-LLP-2012-ES-KA3-KA3MP). Here, we have defined a relatively big and complicated metamodel, based on the complete IMS Learning Design specification, which comprises dozens of concepts and relationships that model the design of units of learning [KO04]. A model instance with a very simple specification generated for a user evaluation is presented in Fig. 5.15. We then asked project partners to compare this instance editor with a Web-based collaborative tool, previously developed specifically for editing IMS Learning Design specifications. As the partners were remotely located across Europe, we have used authoring sessions on the Web using SyncMeta with a parallel activated voice chat and a demonstration video comparing the two tools. 14 participants took part in the evaluation. On average, the attendees had a medium level of expertise on the specification

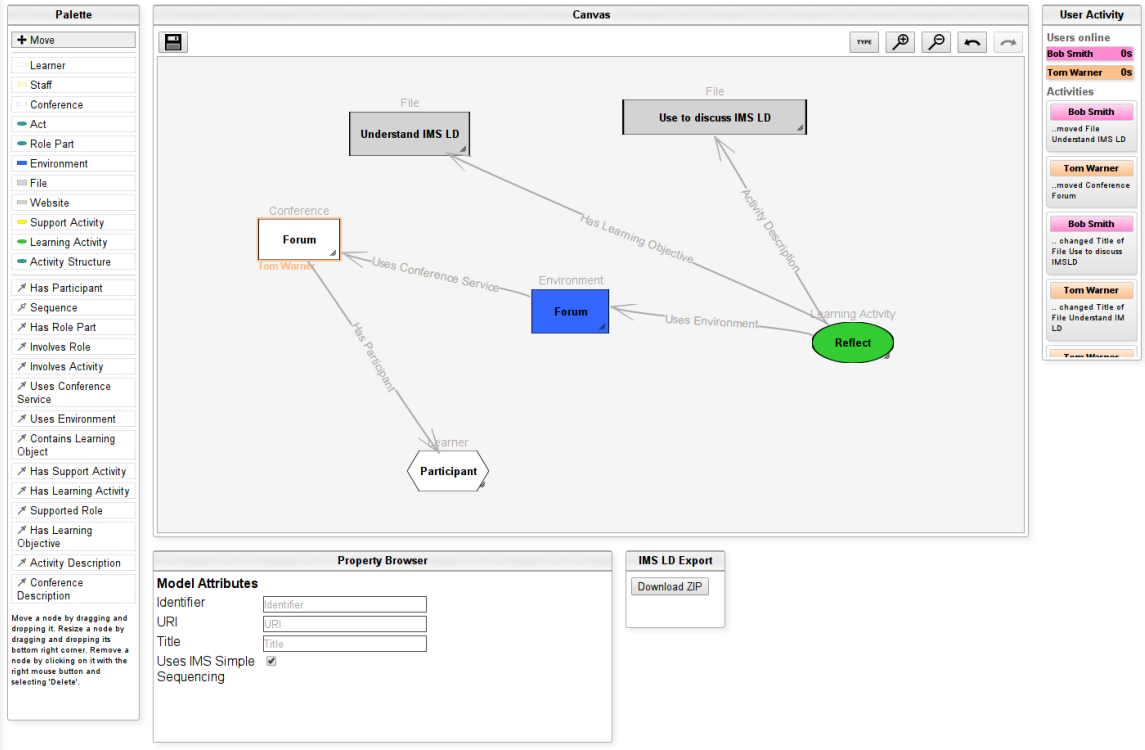


Figure 5.15: A simple IMS Learning Design model used in the user evaluation study

language ($M = 3.93$; $SD = 1.64$). Two videos have been produced, each of them demoing one of the tools. During the presentation, the same model has been created with both editors. Each video was divided into three parts. In each part, a different type of user interaction with the user interface was utilized for the definition of the model. To determine the rating of the evaluators with regard to the user interface and the usability of both tools, the videos have been embedded side by side into a questionnaire, on one page for each part. In all items the editor based on the SyncMeta framework ($M = 5.26$; $SD = 1.67$) has been rated more positive than the other tool ($M = 4.39$; $SD = 1.07$). The answers showed a noticeable tendency towards the editor based on the SyncMeta framework with regard to the user interface, its usability and the general modeling workflow. The participants considered the complete visualization of the model to be a plus of the editor based on SyncMeta. However, regarding scaling in terms of the size of the model, they saw the competing tool to be at an advantage due to its distributed UI, as it would be harder to overview the whole model using SyncMeta's approach. In addition, the awareness functionality of SyncMeta was mentioned as an advantage. As this evaluation was performed without views, a core requirement resulting from this study was to mitigate the complexity of large models using a filtering functionality that only displays nodes of a certain type and hides the others.

5.5.2 Evaluation of View-Based Modeling

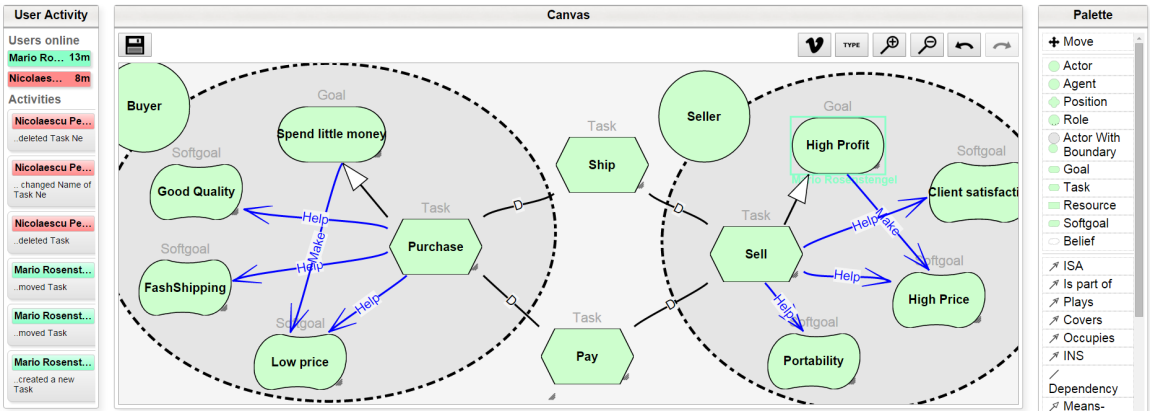
In the second user evaluation, which was performed using SyncMeta with the views extension enabled, the main goal was to evaluate the usability and usefulness of the view-based modeling approach and monitor the NRT collaboration features. Furthermore, we wanted to study the effects which view-based collaborative modeling has on modelers compared to modeling without any enabled views using our SyncMeta implementation and if the time spent for collaborative modeling can be improved by using views.

Participants. The end user evaluation comprised four sessions with four participants each with a total of 16 participants, who were recruited from researchers and students of our department. Their expertise in conceptual modeling, i^* and SyncMeta was measured using seven-point Likert scale (from 1=novice to 7=expert). The results show that users had varying existing knowledge of modeling: expertise with graphical editors was quite high, but has a high standard deviation ($M = 4.38; SD = 2.42$). The same holds for user's general expertise in conceptual modeling ($M = 4.5; SD = 2.46$). The level of expertise with i^* modeling was rather low ($M = 2.44; SD = 2.5$).

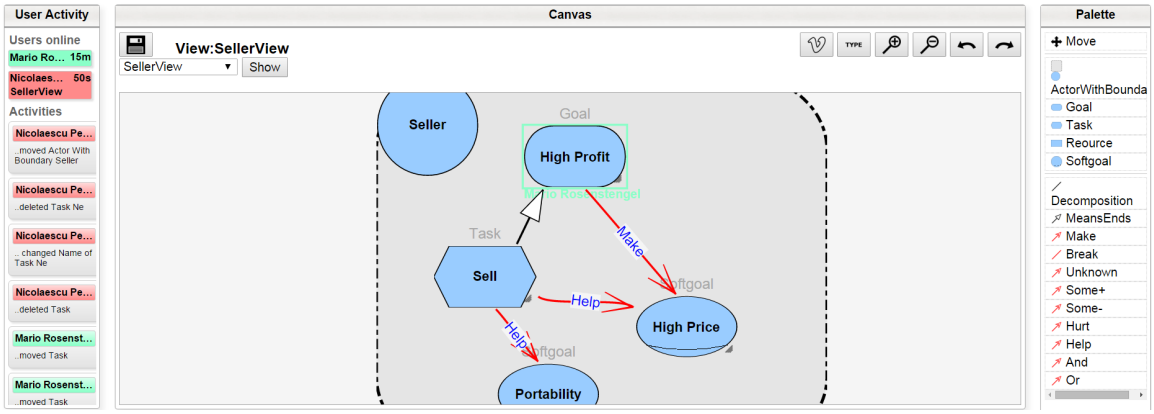
Methodology. In each session, the four participants were split into two groups (*Group Alpha* and *Group Beta*) with two people each. Both groups had to complete two tasks of comparable scope. Each task comprised a list of detailed instructions to extend a given i^* model with additional nodes and edges. This could be performed without any i^* expertise. The collaborators could decide for themselves how to complete the instructions by communicating with each other via chat or just start modeling and let SyncMeta resolve potential conflicts. Fig. 5.16(a) shows a simple i^* model [Yu95] about buyer-seller relationships. Fig. 5.16(b) depicts a possible view on the model, which is called "SellerView" and was generated for the i^* evaluation setting. It contains only nodes and edges within the boundary of the "Seller" actor, along with their edges. For demonstration purposes we also defined a slightly different styling for node and edge types. The first task was solved by Group Alpha and consisted of a predefined view applied, which customized the model editor regarding the requirements of the task (see Fig. 5.16(b)). Group Beta solved the same task without any view on the original model (see Fig. 5.16(a)). For the second task, they switched roles: Group Beta used a predefined view, while Group Alpha solved the task without a view.

After each task the session participants were asked to rate statements regarding their experience with and without views. The ratings were made using a seven-point Likert scale ranging from "strongly disagree" (1) to "strongly agree" (7). During the evaluation the working times for each task and group was recorded to determine whether the views had an impact on the time it took modelers to complete the tasks.

Results. The mean ratings for tasks solved with and without views are plotted as series "view-enabled task" and "view-disabled task", respectively, in Fig. 5.17. For most statements there is little difference between the ratings for view-enabled vs. view-disabled task. We ran paired-sample t-tests for view-enabled vs. view-disabled ratings to identify significant



(a) An i^* model of buyer-seller relationships



(b) Applied *SellerView* to the model above

Figure 5.16: Screenshot of the model editor with and without views enabled

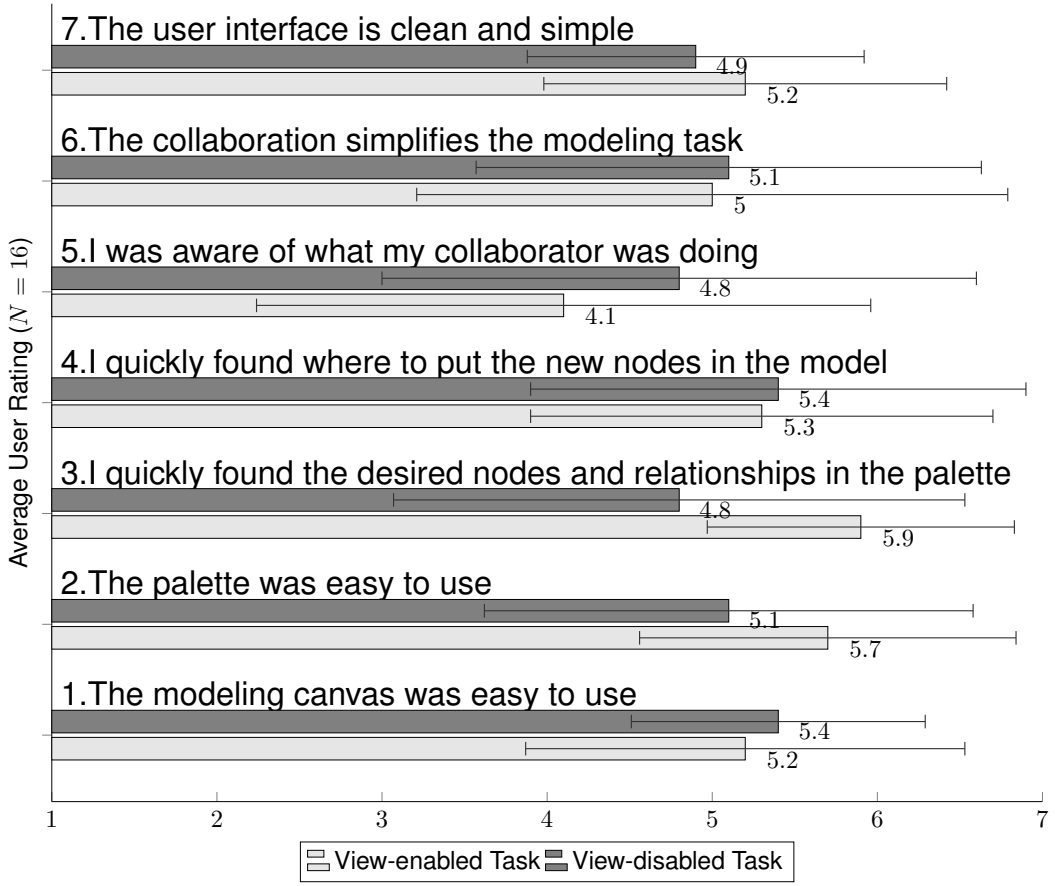


Figure 5.17: Survey results of i^* group sessions (quantitative items).

differences. Two statements exposed significant differences at $p < .05$, namely statement 3, revealing that views helped to find nodes and relationships quicker in the palette ($p = .01$), and statement 5, revealing that the views actually hampered the awareness of the collaborator's edits ($p = .04$).

Additionally the working times for each group and task were recorded. The average working time of Alpha groups for Task 1 without views ($M = 253$ sec.;
$SD = 39$) was on average 82 seconds or 52% longer than the average working time for Task 2 with view enabled ($M = 171$ sec., $SD = 19$). The average working time of the Beta groups for Task 2 without views ($M = 191$ sec.; $SD = 22$) was on average about 24 seconds or 15% longer in comparison to Task 1 with views enabled ($M = 167$ sec.; $SD = 31$) . The lower improvement factor for the Beta groups compared to the Alpha groups can be explained by a learning curve. The Alpha groups used the views during the second task, where they were more familiar with how to work with the tool. Conversely, the Beta groups used the views during the first task when it was the first use with the tool for most of them. This actually shows that modeling with views speeds up the modeling process even for users who are unfamiliar with the tool.

Views can improve user experience can also be used to customize the model editor in order to ease adoption and to improve stakeholder involvement during the collaborative modeling process. Participants also provided some textual comments about the view-based modeling approach. They stated that they liked switching between views and that the reduced palette gives a better orientation, which may explain the faster modeling times with views enabled. In the evaluation, NRT collaboration and edits awareness were only available between views and the entire model editor. However, the evaluation results have shown that users require also collaboration directly between individual views. This feature was implemented as consequence in the latest SyncMeta version using Yjs.

5.5.3 Evaluation of the Nudging Assistance during Modeling

In a last evaluation using SyncMeta with the nudging editor enabled, the goals were to assess the usability of the nudges and the impact of the nudges on the modeling speed and overall user satisfaction.

Participants and Methodology. The evaluation was conducted in eight sessions with two participants per session. In total, 16 participants including researchers and students from our department completed the study. Each session started with an introduction into SyncMeta and a brief summary of the assistance system and its core features. We explained specifically how the palette and the canvas context menu can be used to create nodes and edges and how the guidance strategy can be enabled and used, including the types of guidance and their meaning. We also provided a very short example on how the guidance and awareness features function. After this introductory session, the participants were asked to recreate two simple i^* models (very similar to the one presented in the previous evaluation, cf. Fig. 5.16(a)), provided on paper. The two models were similar in terms of complexity. We have predefined a GMS before the evaluations were performed, containing four activities. The guidance was designed to cover most of the concepts defined in our guidance modeling language (control nodes, actions, create objects and relationships, etc.). The collaborative evaluation sessions took place in the same room. However, modelers could not see each other's screen and were asked to not communicate verbally, such that they can make use of the awareness and guidance features of SyncMeta. In order to distinguish the guidance effects, participants were asked to author one i^* model without any guidance enabled and the second model with the guidance enabled. they were also asked to choose freely one of the two predefined guidance strategies. Finally, the participants were asked to fill out an online questionnaire consisting of Likert scale items ranging from 1 (very bad) to 5 (very good). This included general questions about their background, usefulness and usability of the guidance, guidance strategy, usefulness of the chosen strategy and open questions for problems occurred during modeling, suggestions for improvement, etc.

Results. We conducted a total of 7 sessions and with 14 participants, 7 having already used SyncMeta before. Only few participants were familiar with assistant systems ($N = 4$). The

participants were mostly unfamiliar with the i^* modeling language ($M = 2.36$, $SD = 1.55$). An overview of the evaluation results for some of the main usability and usefulness questions is presented in Fig. 5.18.

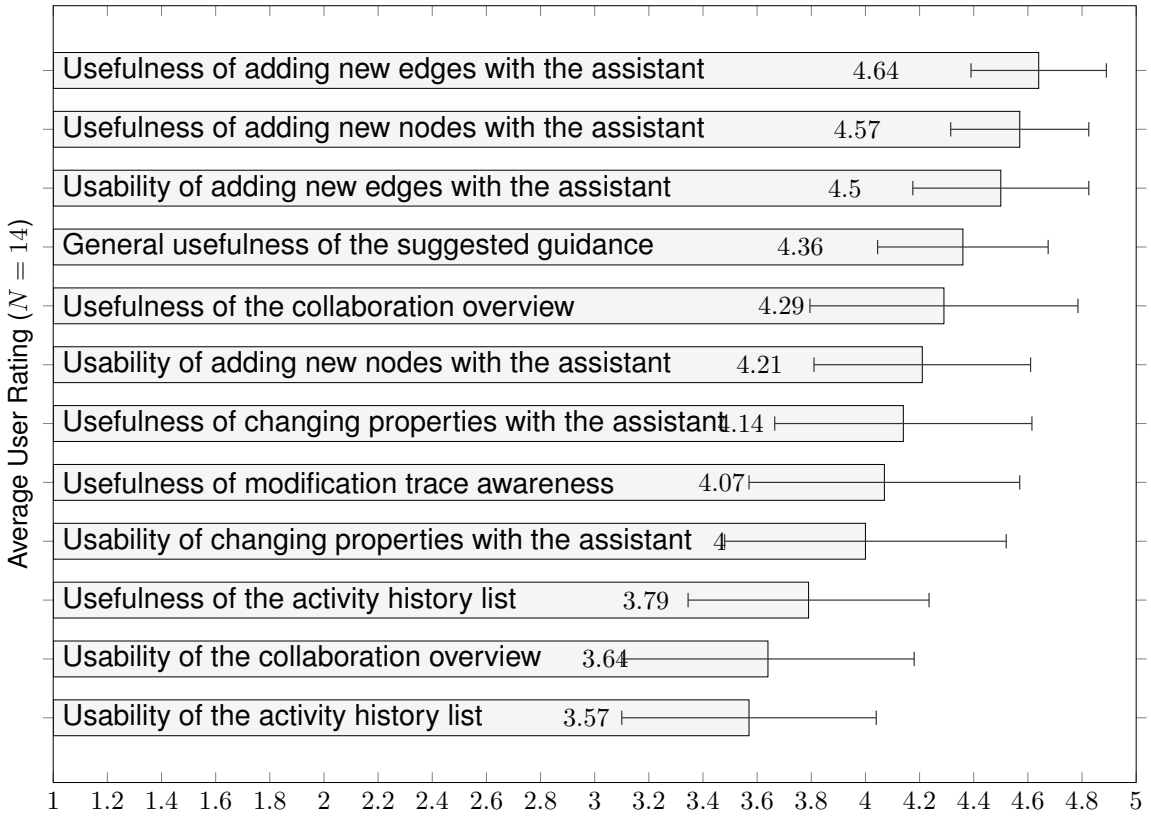


Figure 5.18: Survey results of the end user evaluation.

In general, the guidance assistant was rated positively ($M = 4.18$, $SD = 0.33$). Among the highest rated components in terms of usefulness were the edge guidance ($M = 4.64$, $SD = 0.5$), as this allowed the direct connection of nodes with only 1 click, the tool select guidance which helps to add new nodes ($M = 4.57$, $SD = 0.51$) and the collaboration overview widget ($M = 4.29$, $SD = 0.99$). The edge guidance also got the highest rating regarding its usability ($M = 4.5$, $SD = 0.65$). The activity list received the lowest scores ($M = 3.79$, $SD = 0.89$) as usefulness and usability ($M = 3.57$, $SD = 0.94$) for nudging. A reason for this is its rather static information, as during the evaluation the widget only showcased the names of the most recent activities. We have taken this feedback into consideration by adding time and navigation directly to the canvas element from the activity list. The general usefulness of the suggested guidance during modeling received also a high rating ($M = 4.36$, $SD = 0.63$). The collaboration strategy was the most frequent chosen for the modeling tasks ($N = 9$) and received a positive score ($M = 4.33$, $SD = 0.5$). The avoid conflicts strategy had a similar rating ($M = 4.4$, $SD = 0.89$). Within the open questions, the participants expressed concerns about the guidance usefulness for experienced

users, familiar with the modeling language and the usual tasks. A more elaborate evaluation with novice and experts community users, using SyncMeta for a longer period of time would be needed in order to determine this exactly. Two users stated that the guidance was useful but they would have wanted more guidance during the modeling task. Other improvement suggestion was to make the guidance dependent on selected nodes or edges as well and change the available nudges dynamically upon selecting other entities. The collaboration overview widget can be extended with navigation functionality, such that modelers can also navigate through the canvas directly using this component. Furthermore, two participants suggested to offer navigable nudge items in the nudge widget, augmenting this way the interactions which are currently only possible in the canvas widget. Furthermore, one participant suggested to let modelers define their activities, such that other collaborators are aware directly of their intentions.

5.6 Communication: Discussion and Lessons Learned

This chapter presents the NRT collaborative view-based conceptual modeling framework and its implementation called SyncMeta. We show how the visual view-based metamodeling and metamodel-based view generation can be effectively combined with NRT collaboration during modeling for CoP members with different competences and roles. We present a generic metamodeling approach for visual languages, featuring a metamodel-to-model generation, guidance definition and modeling assistant and the NRT collaboration algorithms and protocols which can be used in order to achieve a flexible and simple collaborative modeling experience in the Web browser. We offer a unique approach of NRT collaboration for view-based conceptual model editing on the Web using optimistic concurrency control mechanisms, combined with known techniques for views definition and generation from information systems domain.

SyncMeta was presented at various international conferences involving Information Systems research communities, such as Modellierung 2014 [DEN+14], ER 2015 [DNE+15], CAiSE 2016 [NRD+16]. The various discussions with established researchers from these communities revealed the high degree of novelty and applicability of our approach in various settings. Apart from the first IMS LD collaborative Web editor, this realizes one of the first Web-based collaborative i^* editors and view-based NRT collaborative conceptual modeling frameworks for CIS design.

The various studies presented above show the feasibility of view-based NRT conceptual collaborative modeling for various scenarios. Views enable customization in specific domains and simplify understanding of models for target stakeholder groups. The results also show that the views are useful for reducing complexity, especially when dealing with large models. In NRT settings, they improve usability and efficiency during modeling but need to be also sustained via awareness of modeler's actions and relevant nudging strategies, especially for novices when dealing with no constraints introduced in the collaboration

process. Among its multiple usages, we show SyncMeta's feasibility for agile model-driven Web information systems modeling, in order to target special needs and preferences of multiple stakeholders during the software engineering processes.

As described in [DNE+15], the visual-based (meta)modeling approach of SyncMeta has some restrictions, such as model checking functionalities on the (meta)modeling layer. By extension, the views do not allow the specification of cardinalities or multiplicities with regard to the relationships and view-relationships. Furthermore, SyncMeta does not support any model checking functionalities either on the metamodeling or the modeling layer. Also, it is not possible to define conditions on inherited attributes of superclasses. In the current implementation, only the definition of conditions for the attributes of the referenced class is allowed. A simple solution for this problem is that we define the attribute directly in the referenced class, but this is suboptimal and fails to exploit the inheritance hierarchy. Finally, SyncMeta does not support yet sophisticated transformations like aggregations on meta classes and non-derived relations between them. Such features are conceivable for our NRT collaborative approach. The major task would be to keep model and view in a consistent state and ensure correct transformations in both directions.

All in all, the various evaluations, usage and performed user studies show the real need of NRT collaboration during modeling and the multiple benefits in terms of simplicity and usability offered by this approach. The scalability of P2P collaboration is ensured by integrating the Yjs library as a powerful shared editing facilitator. Based on the proven reliability and performance of the Yjs library, also due to its adoption in various Web-based projects, SyncMeta can scale up to dozens of users without significant performance degradation on the collaboration level of the framework. Further usage and evaluation studies need to determine whether such numbers are relevant and can be supported by the view-based model editor implementation and how usable the framework —with views enabled— is when dealing such numbers of modelers. We consider that the performed studies showcase the power of SyncMeta and offer a glimpse into the opportunities opened through our collaborative Web-based approach for agile collaborative modeling within our MDWE conceptual and technical framework.

Our first intuitions are the true ones.

Emile M. Cioran (1911 - 1995)

Chapter 6

Community Application Editor Validation

Summary This chapter contains the remaining steps of our design science approach concerning the realization of CAE. As seen in Fig. 6.1, depicted as gray areas, a major part of the conceptual considerations of CAE were explained in Chapter 3. Here, we continue with the main architectural design considerations and we detail the relationship of our approach to the ATLAS CIS model. Then, as the Yjs and SyncMeta contributions are already explained in the previous chapters, we then present the technical realization of CAE and report on the studies performed for the framework's evaluation. **Contribution** RQ1, RQ2. *Keywords:* MDWE; NRT Shared Code Editing; Model to Code Synchronization; Open Source; GitHub; Docker; Microservice Architecture; SyncMeta; las2peer; Inter-widget Communication; Yjs. The results presented here have been published in [NRD+18, dNKJ17, dND+16]. This section contains partial information and content extracted from these publications.

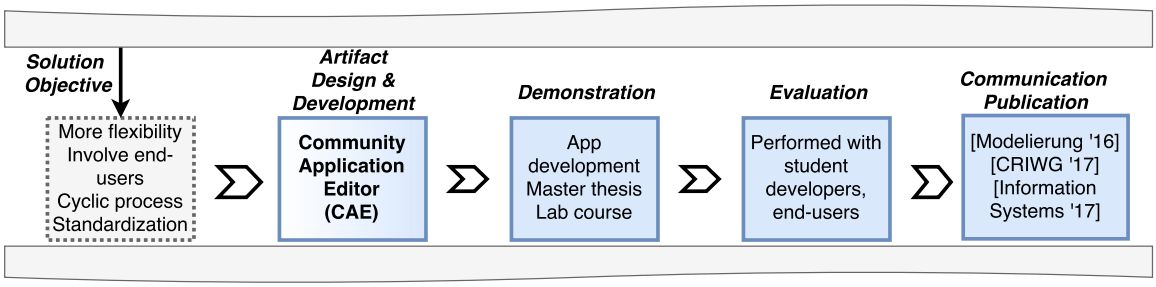


Figure 6.1: Design science approach: Community Application Editor realization and validation

6.1 Design Considerations

CAE is developed on top of the SyncMeta framework, as this offers the core NRT collaborative modeling functionality and is realized to fulfill the CAE requirements and abstract them for future customizations and usage. Furthermore, it also relies on Yjs for the NRT coding and model-to-code synchronization implementation, as well as for adding NRT shared editing features to the generated applications. Overall, CAE respects the following design considerations:

Widgetized prototype The Community Application Editor – same as SyncMeta – features a widget-based frontend, adhering to the core principles of our widgetizing methodology. Using the same look and feel as the generated frontends is meant to ease CAE's usability and increase the compatibility with future CAE, SyncMeta and other Web widgets developed by communities

Microservice architecture Since CAE needs only limited server-side logic, we have split the core functionality into two microservices, that handle the persistence of models and the code generation accordingly

DevOpsUse-ready Since we support the automatic preview and deployment of widgets and microservices, CAE uses important technologies part of the DevOpsUse [dNKK16, RKKJ17] community-centric development methodology, such as Jenkins and docker for continuous integration and automatic, container-based deployment. Both CAE and the resulting applications are by default open source on GitHub

Communication exchange support In its implementation, CAE realizes all four communication and message exchange mediums described by the concept. It supports P2P message exchange on the backend and Yjs-enabled collaboration on the frontend, it features IWC using the ROLE SDK, Yjs compatible IWC library and finally, it allows frontend-to-backend and backend-to-backend communication via RESTful calls.

6.2 ATLAS-based CIS Lifecycle Approach

Conducting all phases of a research methodology in a CoP is a rather difficult task. Researchers need to cope with community and project lifecycles, access gain to the practice and most important understand and be part of the community to be studied. Therefore, we have used during this work existing developer communities and tools established at our department and within the research scope of our group through participation and organization of academic and community events. Moreover, we have used CIS testbeds comprising not only researchers, but also developers, end users and domain experts from various fields, engaged in professional CoPs, with whom we were mostly collaborating in various projects, mentioned in the introduction of this work.

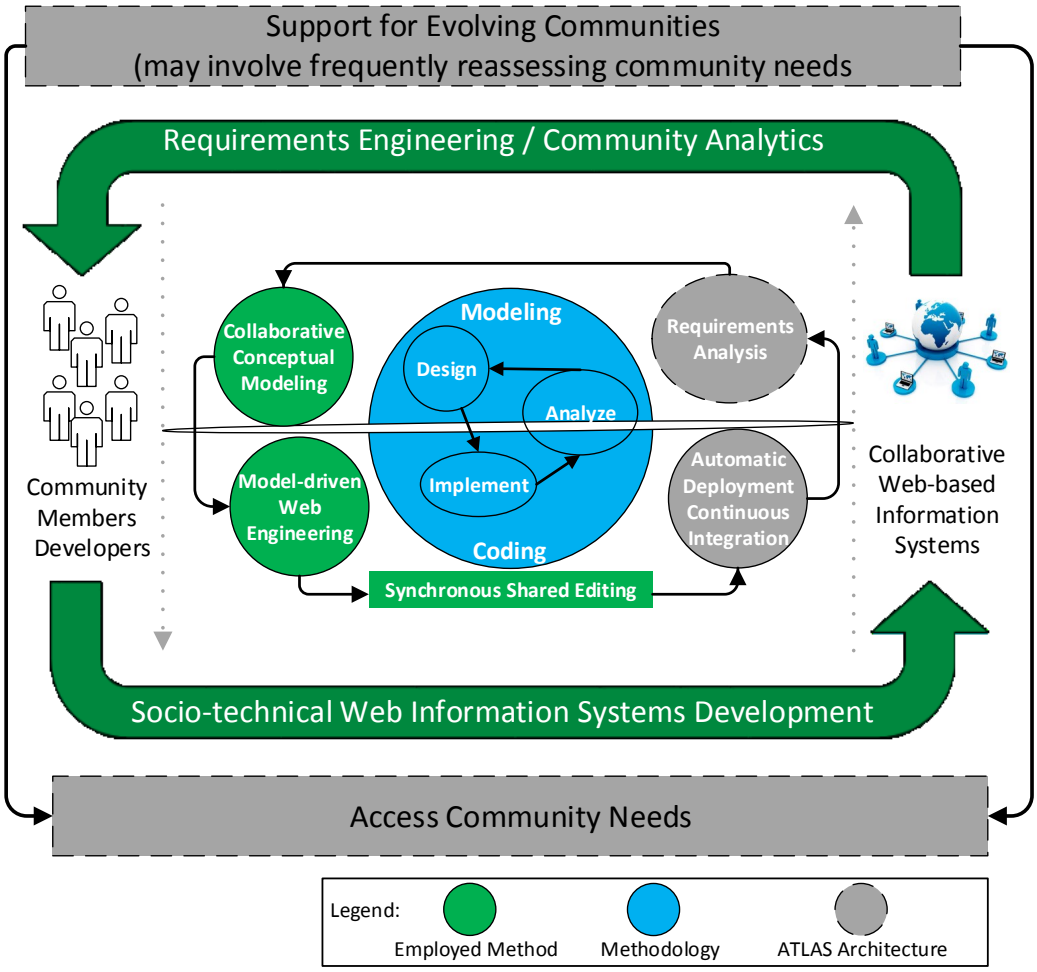


Figure 6.2: Approach based on the the ATLAS/DevOpsUse methodology

Fig. 6.2 presents the integration of our approach into the ATLAS methodology. The gray rectangles represent the core elements of ATLAS, that correspond to accessing community needs and offering support for evolving CoPs. In our approach, this concept is specialized by offering support for community members and developers (focusing on the latter) for engineering collaborative Web-based CIS. This is achieved through a cyclic development approach, involving modeling and coding and three main steps, “Design”, “Implement” and “Analyze”.

Using existing methods such as DevOpsUse (cf. Fig. 2.6), social requirements analysis, automatic deployment and continuous integration and adding the core contributions of this dissertation (collaborative conceptual modeling, MDWE and synchronous shared editing methods), we provide the link between our stakeholders and the CIS by means of socio-

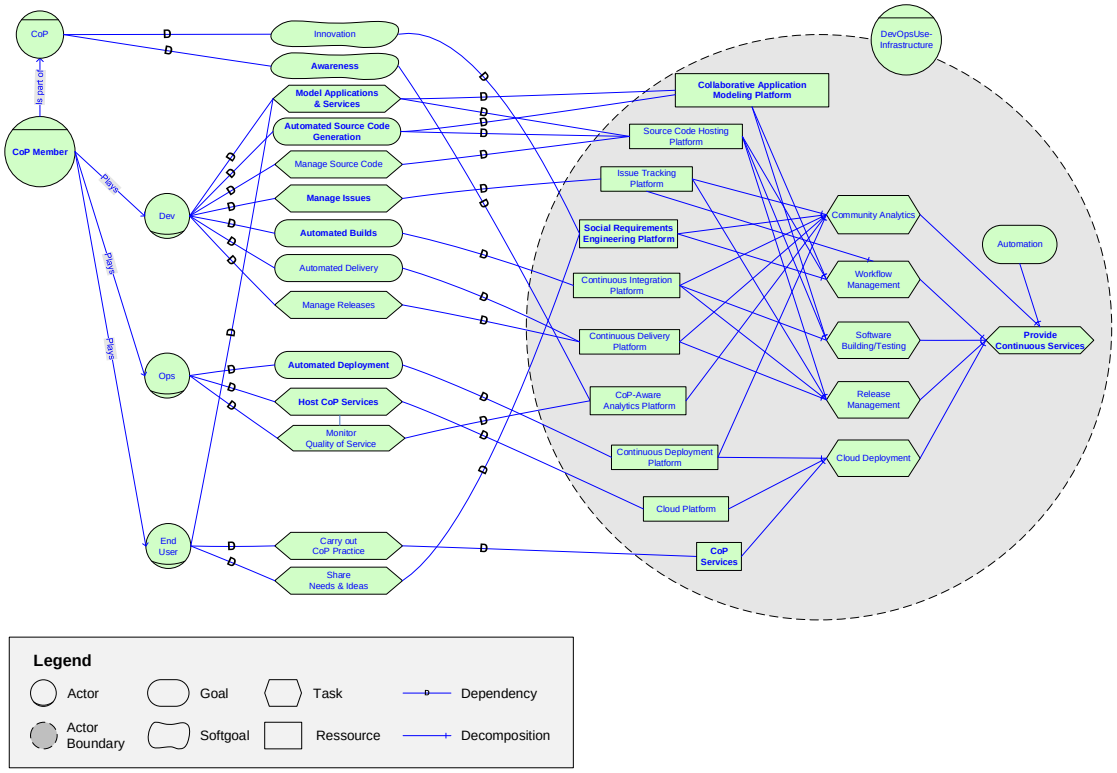


Figure 6.3: Detailed view on the integration with the DevOpsUse methodology [KKN+15]

technical Web CIS development and requirements engineering. A work in progress that tries to create a formal representation of DevOpsUse [KKN+15], developed by Renzel, Koren et al. [RKKJ17], using an $i*$ notation, is depicted in Fig. 6.3. $i*$ [Yu97] is an abstract, high-level, agent and goal-oriented modeling language that can be used for requirements engineering and can express dependencies (such as goals, tasks, resources) between arbitrary, not necessarily human actors. One can observe that the main actors are the community (CoP), its members having various roles and the DevOpsUse infrastructure. The matching between DevOpsUse and our contributions is highlighted in bold. We use awareness and NRT collaboration processes to support CoP goals. Developers are supported in building the CIS infrastructure through the modeling of applications and services, automated source code generation and automated builds, by means of a platform for collaborative application modeling and using a social requirements engineering tool. The developed services can also be automatically deployed and hosted such that we achieve agility in CIS engineering, offering continuous service design, development and deployment capabilities in NRT and involving end users in this process.

6.3 System Overview

The architecture of CAE is depicted in Fig. 6.4. CAE uses the same concepts and technologies as its generated applications, relying on widgets and microservices.

The backend consists of two microservices, which contain the logic for model storage and code generation. The microservices are implemented using las2peer [KRdJ16], a Java-based open source framework for distributing community services in a peer-to-peer (P2P) infrastructure. This offers a highly reliable and secure platform for hosting microservice backends, supporting end-to-end encrypted communication and storage facilities as well as load balancing across servers. The microservices communicate with each other using the P2P network communication functionality. Network-wide (and thus cross-service) the user and group management are directly supported by the las2peer API. Moreover, by using the widget space provided by the ROLE SDK as frontend container, we achieve a unified login using the OpenId Connect standard. The models are extracted from the SyncMeta Yjs persistence and stored in a dedicated database. The code generation microservice uses git to push the generated code into dedicated repositories. As by default all applications are open source, we use a dedicated GitHub organization for the code. The code generation microservice uses predefined templates, also stored on GitHub, that contain skeletons for widgets and a las2peer microservices. The skeletons are copied, the code generated from the model applied over the default code and the resulting elements committed into a new repository. After this step, the code can be refined and any model or code changes updated live. We integrated the model synchronization and trace generation functionality into the code generation microservice as well. This can manage local Git repositories used by the live code editor widget in order to update the source code. The live code editor is realized within a "Live Code Editor" widget based on the ACE editor. Integrated with the code editor, our framework only needs to manage the Yjs collaboration spaces (similar to a chat room, with all involved users being able to collaborate on one application model and code). Frontend to backend communication is done via RESTful service calls. For this purpose we use las2peer's RESTful interface, which is offered using a Web connector. The communication between widgets is achieved through IWC, offered by the ROLE SDK[1]. On the frontend, the generated applications support IWC communication, who-is-online awareness information and a widget space chat, offered by the ROLE SDK and the SyncMeta framework. NRT collaboration on HTML5 elements is also provided by default and can be modeled in the frontend view. For this purpose, CAE uses Yjs, by automatically adding the library to the frontend component and binding it to the corresponding elements.

Fig. 6.5 depicts our approach to view-based model-driven Web engineering, based on the initial scenario. Fig. 6.5(a) depicts a Web information system consisting of a frontend component realized as a widget and a backend component realized as a microservice. The widget consists of several HTML elements associated with functions and events. The microservice provides four HTTP methods together with a relational database. Fig. 6.5(b)

[1] http://www.role-project.eu

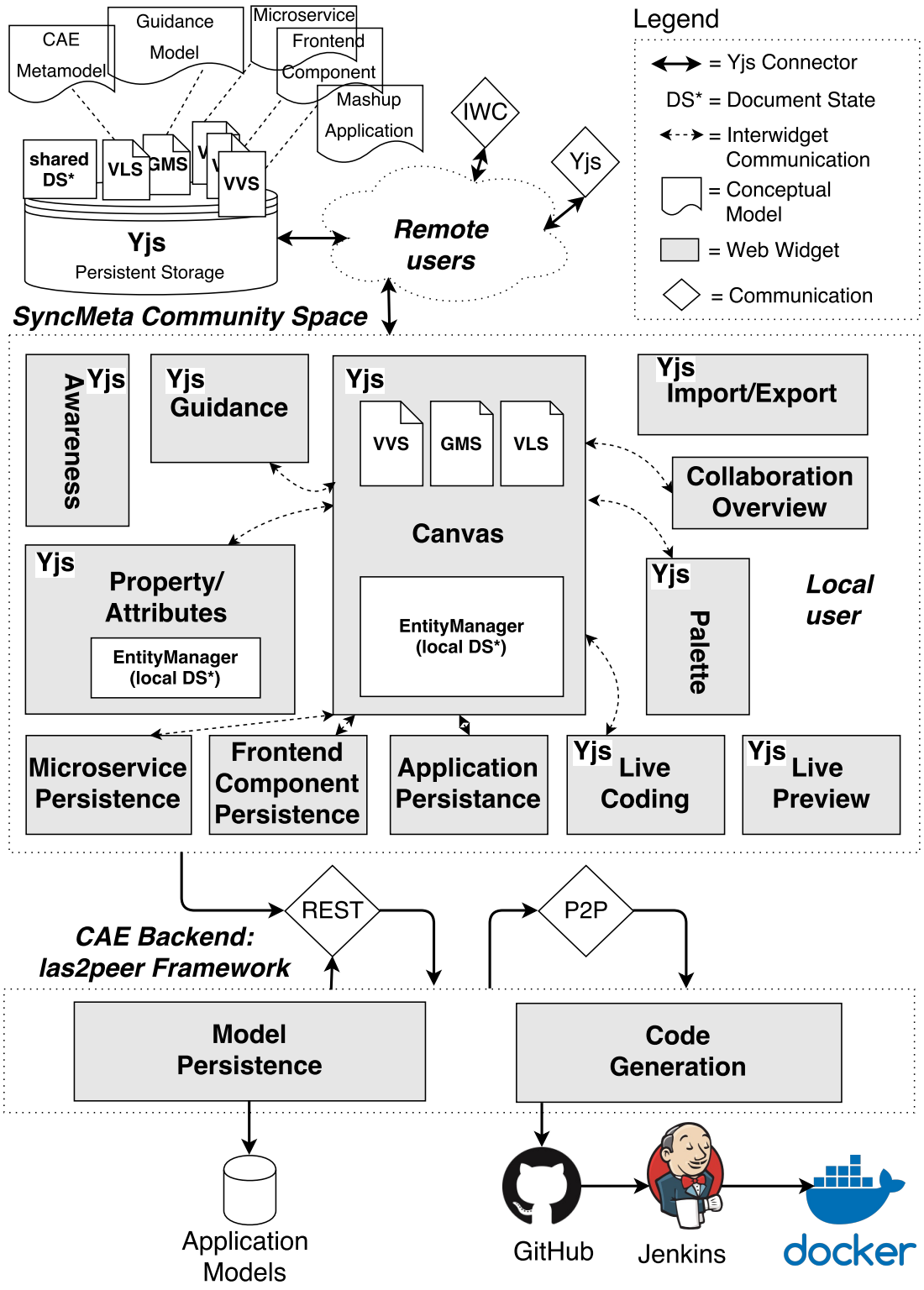
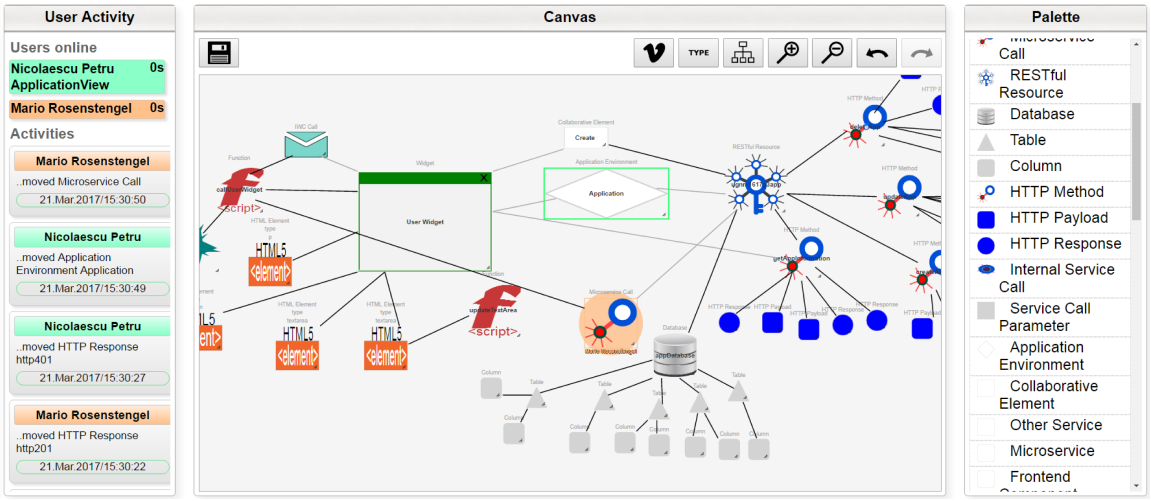
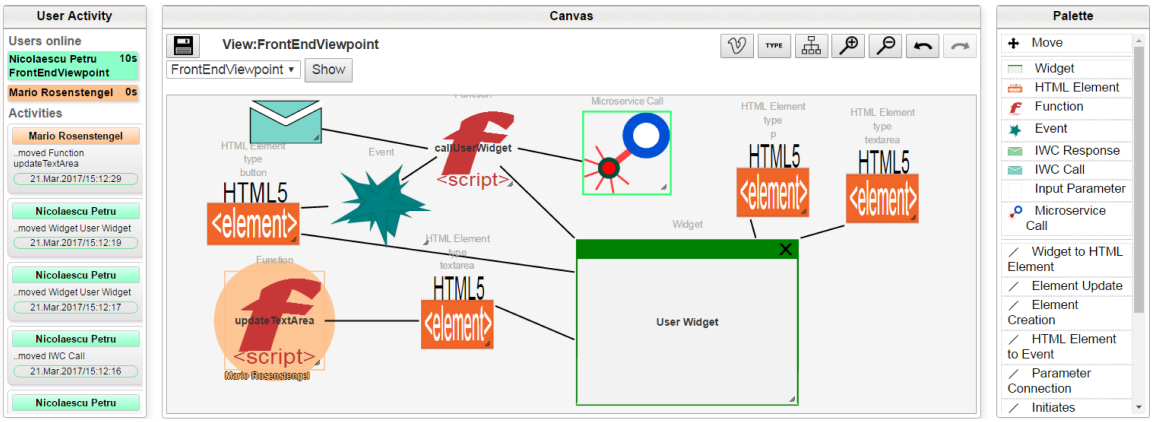


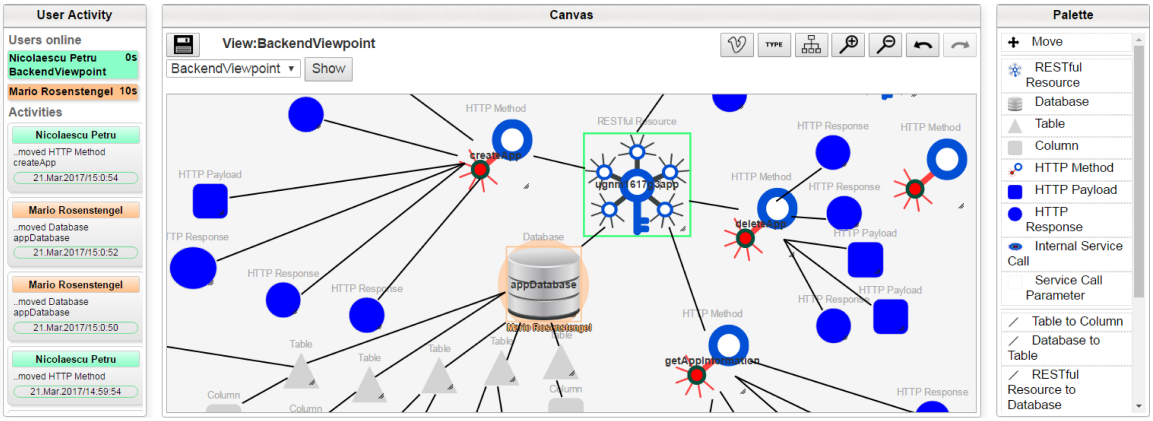
Figure 6.4: Community Application Editor architecture overview



(a) Community Application Model



(b) Applied *FrontendViewpoint* to the model above



(c) Applied *BackendViewpoint* to the model above

Figure 6.5: Screenshot of the model editor with and without views enabled [NRD+18]

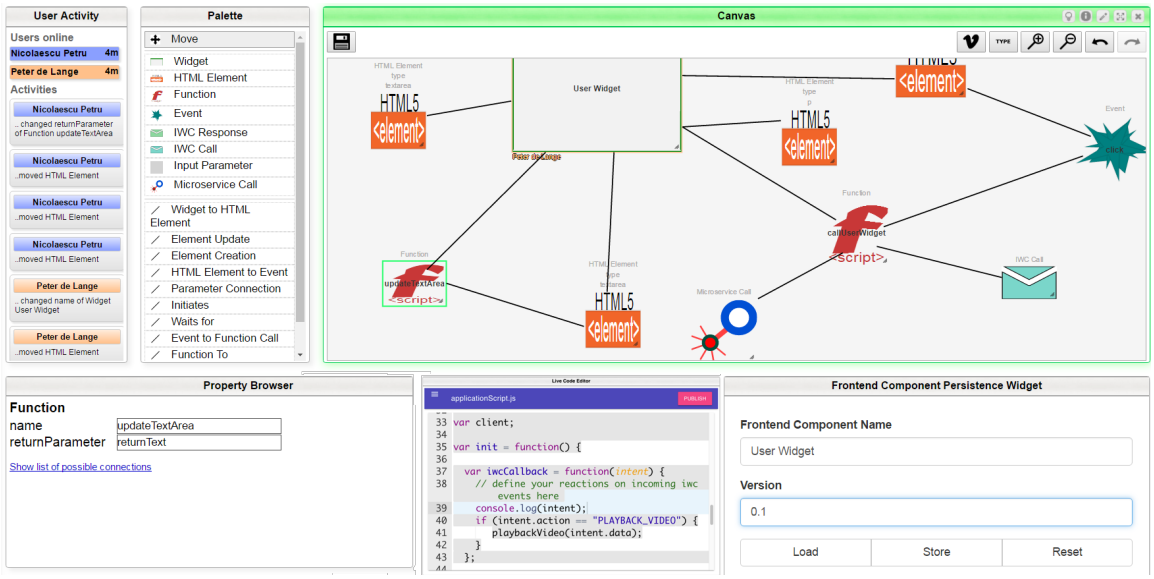


Figure 6.6: Frontend component view screenshot of a user widget

and Fig. 6.5(c) depict two possible views on our system architecture. The "BackendView"(cf. Fig. 6.5(c)) depicts only the microservices of the system. Analogously we defined a viewpoint for the corresponding frontend component view depicted in Fig. 6.5(b). End users without technical knowledge can collaboratively create the frontends they require on the frontend view, together with developers. Awareness information is depicted in the modeling canvas and "User Activity" widget. For a given model, each modeler can see in which views other users are working on and which element is currently being edited and by which user. This information is updated in NRT as text in the user activity list and also displayed on the canvas, by adding the modeler's name and highlighting in a user color the node or edge which is currently being authored. Awareness information about the latest nodes visited by a certain modeler is also highlighted as a circle in the user color. The color fades with time and offers a temporary visual trace of the actions performed by a certain user. Using this awareness information and other communication means (chat, video, etc.), developers can give immediate feedback on the required functionality. Developers can edit in NRT the application backend and the communication between the backend and frontend, while architects can follow in NRT the creation of the overall model and check the integrity of the modeled system. The palette widgets adapt to the view by only displaying node and edge types defined in viewpoints. As such, the view-based modeling approach allows developers to focus on their specific field of expertise.

To give a better overview on how CAE's application modeling and code generation processes work in practice, we describe here an example setting of how a widget depicting user management functions is modeled and generated (cf. Fig. 6.6). The first step is the collaborative modeling process in the frontend component view. The community uses

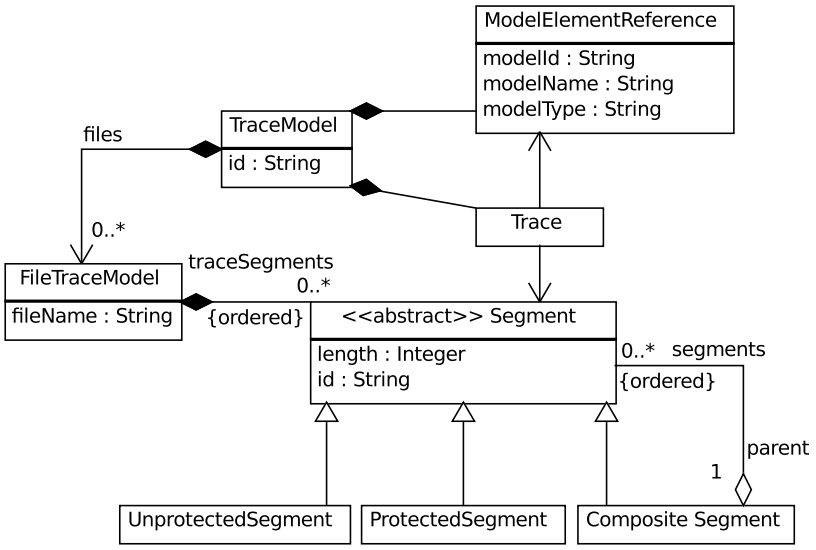


Figure 6.7: Trace model

the *Palette* widget of SyncMeta to add nodes and edges to the modeling *Canvas* widget, according to the predefined frontend component metamodel, which prevents for example the insertion of wrong edges. When the modeling process is done, one of the community members uses the *Frontend Component Persistence Widget* to store the model. The store functionality of the widget invokes the *Model Persistence Service*, one of the two backend services. This service then parses the passed SyncMeta model and offers CRUD operations using the relational database. Its second responsibility is to invoke the *Code Generation Service*, which is responsible for the model to code transformation. This process is based on predefined templates available on GitHub, that get modified via code injection according to the passed model. The generated code is then pushed to a newly created GitHub repository. The deployment of a complete application is done in the mashup view. After all needed microservices and frontend components are added to the canvas, the complete application gets generated from the single components and is put into a Docker[2] container which both starts the backend microservice network as well as a server with the frontend components.

Trace Generation and Model Synchronization The template engine, implemented in the backend of the CAE, forms the main component for trace generation and model synchronization. It is used for both the initial code generation as well as for further model synchronization processes. Except for some special cases, like renaming or deleting files, applying the strategy design pattern allows us to use the same methods for both the initial code generation and model synchronization.

Fig. 6.7 depicts our trace model, adapted from the metamodel of traces presented in [OO07]. For each *FileTraceModel*, and thus for each file, we instantiate a template engine class, which can hold several template objects. A template object is a composition of *Segments*,

[2] `https://www.docker.com`

generated by parsing a template file. A template file contains the basic structure of an element of the Web application's metamodel. In such a file, variables are defined to be used as placeholders, which are later replaced with their final values from the model. Additionally, the template file contains information about which part of the generated source code is protected or unprotected. Based on the template syntax for variables and unprotected parts, a template file is parsed and transformed into a composition of segments of the trace model. For each variable a protected segment is added, and for each unprotected part, an unprotected segment is added to the composition. The parts of a template file that are neither variables nor unprotected parts, are also added to the composition as protected segments. According to the definition of model synchronization for M2T transformations, Eq. 3.1 must hold for the model synchronization. Thus, we need to update the content of each variable for all templates of all model elements. However, maintaining a trace and a model element reference for all of these variables is not feasible due to the large size of such a file trace model. Instead, traces are only explicitly maintained for the composition of segments of a template. Linking a segment of a variable to its model element is done implicitly by using the element's id as a prefix for its segment id. When templates are appended to a variable, the type of its linked segment is changed to a composition.

Following the strategy design pattern, we implemented an *Initial Generation Strategy* and a *Synchronization Strategy*, which are used by our template engine. Each synchronization strategy instance holds a reference to the file trace model of the last synchronized source code to detect new model elements as well as to find source code artifacts of updated model elements. As in some cases source code artifacts of model elements can be located in different files, a synchronization strategy can also hold multiple file trace models in order to find code artifacts across files. After a template engine and its template strategy were properly initiated, the template engine is passed as an argument to the code generators. These create template instances for the model elements based on the template engine. The engine checks if a segment of the model element is contained in the trace models file by recursively traversing its segments. If a corresponding segment for the model element was found, a template reusing this segment is returned. Otherwise, a new composition of segments that is obtained by parsing the template file of the model element is used for the returned template. For new model elements, new source code artifacts are generated. For updated elements, their corresponding artifacts are reused and updated. As templates can contain other templates in their variables, these nested templates need to be synchronized as well. In the generated final files, source code artifacts of model elements that were deleted in the updated model must be removed from the source code. Therefore, the nested templates, more specifically their segment compositions, are replaced with special segments by the synchronization strategy. By following the *proxy design pattern*, these special segments are used as proxies for the original compositions and ensure that templates of deleted model elements are removed from the final source code.

To ensure that source code artifacts that directly correspond to a model element are not manually added by users (and thus hold no corresponding modeling element, making the

Figure 6.8: Screenshot of the live code editor widget

model an inaccurate representation of the source code), we implemented a model violation detection. For each detected violation a feedback note containing the position of the violation, as well as a message describing it, is provided to the user. Model violations are defined in terms of violation rules. A violation rule consists of a model element type, a regular expression that is used to find the violation, a group number that can be used to reference a specific group of the regular expression and a message that describes the violation.

Live Code Editor The live code editor allows multiple users to collaboratively work on the same file at the same time. As it can be seen in Fig. 6.8, the code editor widget is divided into three parts. On the left side of the widget a list of currently active users as well as a file list is displayed. The actual editor is located in the center of the code editor widget. The cursors of remote users are displayed in different colors to all users participating in this live coding phase. For highlighting protected segments in the viewport of the Ace editor, we use a gray background color. The depicted screenshot shows the development of a frontend component. Here, a tree containing the widget's *HTML elements* is shown on the right side of the editor. The synchronization of the file content among all users, the synchronization between the source code and its traces and the concept of (un)protected segments are integrated into the code editor. While the content of an unprotected segment can be edited, a protected segment is immutable. In order to synchronize the file content among all users, each unprotected segment is individually synchronized on the frontend, using the previously mentioned Yjs library. As protected segments are not editable, they are not synchronized. Because every unprotected segment is synchronized individually, the length of each segment is also synchronized. Thereby, the transformation of source code changes to updates of the trace model defined in Eq. 3.7 is implicitly performed for all users. Thus, the trace model and source code are implicitly synchronized for all users. We distinguish between navigation, deletion, insertion and other allowed operations. In order to decide to which segment a source code change belongs, a reference to the current active segment is updated in a navigation decorator, every time the local user's text cursor changes.

Based on the active segment, it is decided if a source code change is allowed or forbidden, meaning if the operation is performed in an unprotected or protected segment. A deletion operation is performed if the current active segment is not protected and the dimension of the selected text that should be deleted is not out of the bounds of the active segment. Similar checks are performed for insertion operations. The last group of operations consists of commands that do not change the source code and that can be executed without side effects towards the trace model. On the backend, we extended the CAE's REST API with means for maintaining local Git repositories. When a request for storing a file is received, its content and its file traces are stored and committed to a local repository. Before that, a check is performed to determine if storing the file is allowed. To prevent conflicts with files that are artifacts of an earlier model synchronization process, the id of the trace model that is updated in each model synchronization process is also included in each file trace model. Thus, each file is assigned to the id of the model synchronization process in which it was created. Even if our frontend is designed to reload files after a model synchronization process, this protection mechanism additionally ensures that the synchronization between files and their models does not accidentally break. Possible conflicts with the remote repository are automatically resolved by using the Git *Theirs* merging strategy.

Next, we present the user evaluation studies performed to gather feedback on our cyclic, agile synchronous approach.

6.4 Evaluation using User Studies

We evaluated our approach in two separate studies. The first one[3] was performed in a simulated community setting with mixed teams of developers and non-developers. The second investigated the relevance of our model-code synchronization approach and was conducted with developers.

6.4.1 Evaluation with Heterogenous Teams

Methodology We considered groups of two or three people with various technical backgrounds. We carried out thirteen user evaluation sessions, with a total number of thirty six participants. The groups consisted of at least one experienced Web developer and at least one member without any technical experience in Web development who received a written description of the application to be designed. During the evaluation sessions, the non-technical members had to discuss the application to be developed with the developer team. In general all developers had reasonable experience with Web service development and frontend technologies. After the evaluation session, we conducted oral interviews with the participants. They were also asked to fill in a questionnaire about their experience using CAE.

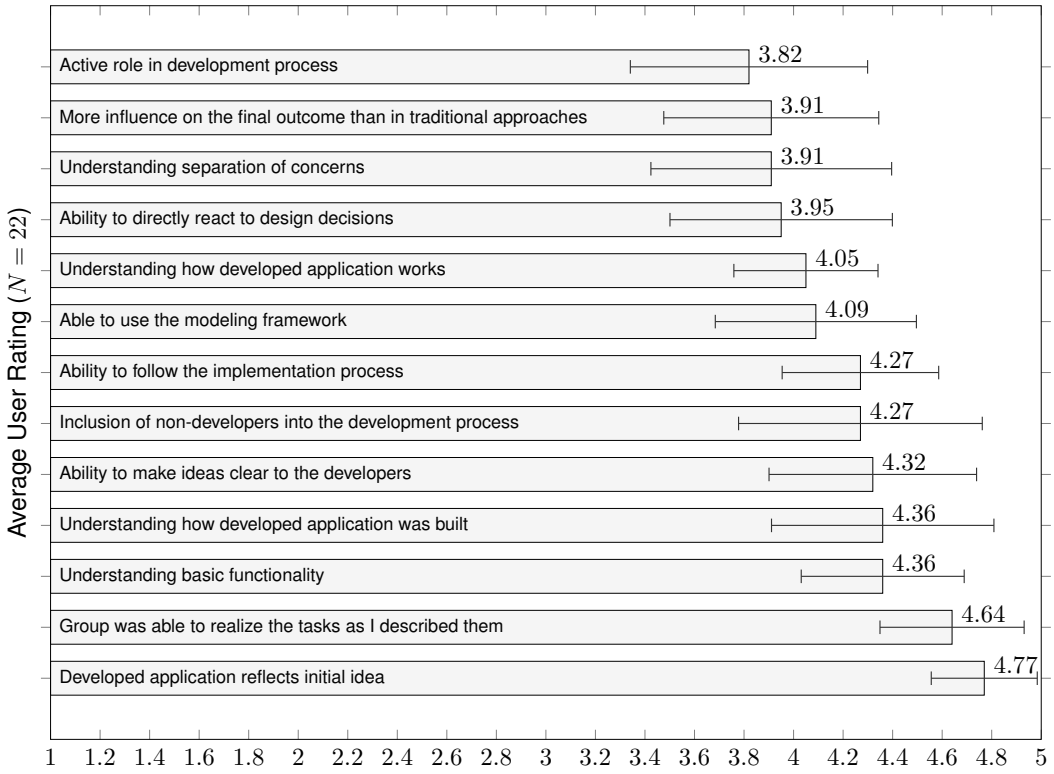


Figure 6.9: Survey results of non-developers

Results The main outcomes of the user study are presented in Fig. 6.9 and Fig. 6.10 In general, we received high ratings from non-developers in terms of methodology. Developers felt they were able to implement the requirements formulated by the non-developers. Most non-developers felt integrated well into the NRT development process and the oral interviews revealed that they could follow the development process well. Although the question if non-developers took an active role in the development process received the lowest score, the result is still satisfactory (3.82 out of 5). From the developer survey, we received the highest ratings for questions regarding CAE's overall applicability and its usability. the collaborative aspects of the framework were also rated rather high by both groups. The oral interviews revealed that most developers felt both the need for requirement analysis improvements regarding the inclusion of non-technical stakeholders as well as that the CAE can be used for this purpose. Most developers recommended a better integration of non-technical users into the development cycle. Among the open questions results, many also considered the reusability of already existing microservices and frontend components in new applications as an important concept. A further benefit highlighted during the evaluation was the opportunity of novice developers to get familiar with a new project by studying an existing model.

[3]the contents of this section have been published in [dNKJ17, de 15]

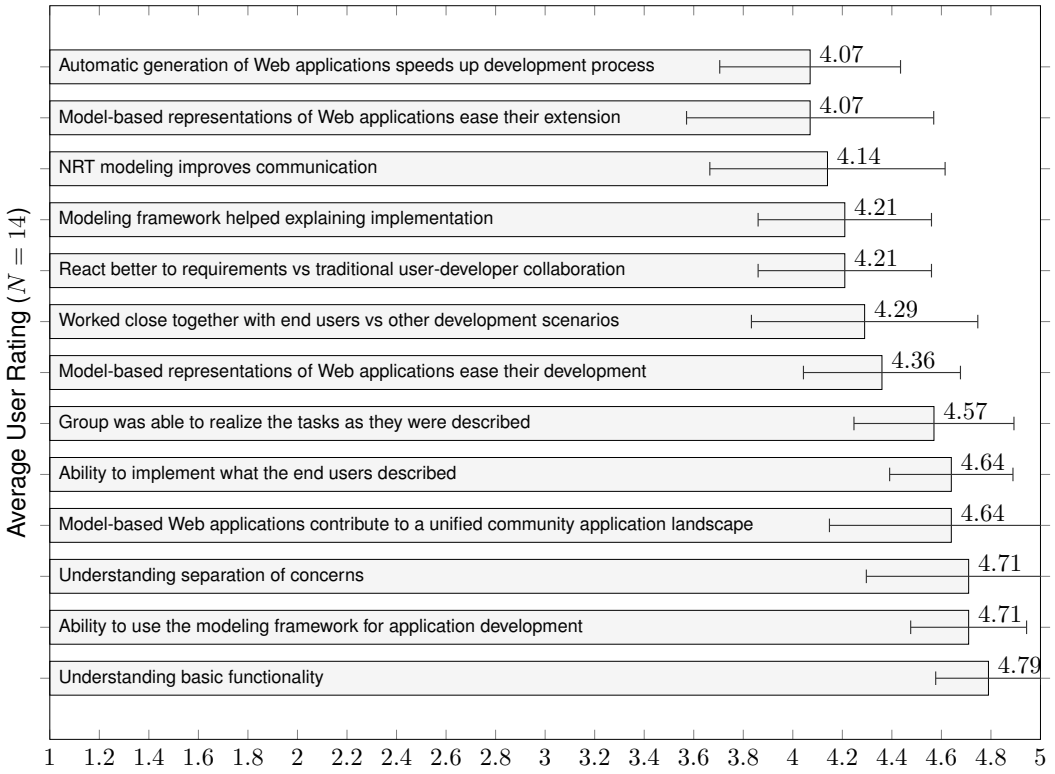


Figure 6.10: Survey results of developers

Concerning proposals for improvement, the most important mention in both groups was that the key aspect for a successful collaborative usage is the improved communication via the NRT approach and awareness of others' actions. A particularly often requested feature was the introduction of a second abstraction tier for the frontend component view, which could hide too technical aspects from non-developers, concentrating more on the "visible" elements, putting the functionality into a second component view, which would then be used by the developers only. The evaluation showed the usefulness of the CAE to integrate non-developers better into the development process. Developers saw the benefit of CAE's MDWE approach to contribute to a unified community application landscape forming the community information system. Among the identified challenges, developers mentioned the possible large size of models for certain applications that can lead to difficulties in browsing the models. Furthermore, participants noticed that even though the modeling language is rather simple, some concepts are still too technical for non-developers. In development terms, the limitation to a specific architectural style and Web development languages were also mentioned as drawbacks. Since CAE was overall well rated by developers (4.40), this can be regarded also as a positive side, since it can support best practices in componentized, widget-based CIS development and has been developed based on feedback from the developer community active in this area. Another point expressed in the answers was the versioning and the awareness, also in offline settings. Although SyncMeta already

offers support in this regard through the activity list widget, combining offline and online shared editing of CIS together with versioning systems and more complicated awareness features are a research direction worth considering. An interesting developer suggestion was to introduce a code versioning extension, for example use different branches on GitHub for different changes and updates performed on the model. All in all, results indicate that designing a microservice architecture through MDWE takes a right step into raising the quality of a modern community information systems engineering process.

6.4.2 Evaluation of the NRT Model Code Synchronization Features

We performed a usability study with student developers to assess how our collaborative MDWE method is received in practice. We carried out eight user evaluation sessions, each consisting of two participants experienced in Web development. After receiving a short introduction into the CAE, the participants were seated in the same room and asked to extend an existing application, which consisted of two frontend components and two corresponding microservices. Each evaluation session took about half an hour of development time. At the end of each session, we let the participants fill out a five Likert scale questionnaire containing questions about their Web development experience and gathered their feedback regarding the cyclic development process and the live code editor.

Results and Observations. As expected, the rating of the familiarity with Web technologies (4.00) and RESTful Web services (4.07) was rather high. However, only a minority of our participants were familiar with MDWE (2.67) or had used collaborative coding for creating Web applications before (2.40).

Fig. 6.11 shows the main results regarding our development paradigm. As it can be observed, the participants rated connections between our two collaborative phases, namely the access to the code editor from the model (4.67) and the reverse process with the synchronization enabled (4.40) very high. The same also holds for the awareness introduced into the live code editor, as the participants could easily see where other developers were working (4.33). They were able to successfully collaborate on a shared part of the application by using the live code editor (4.47). These two rather high ratings show that the developed code editor fulfills the requirements for live collaborative code editing of model-based applications. Compared to the other ratings, the general idea of a collaborative code editor for development and the need for collaboration during the code refinements phase were rated lower (both 3.47). One explanation for this is that developers are familiar with using version control systems and therefore do not see a high demand for a live collaborative code editor when working together with other developers. Even though the chosen application was quite simple and due to time constraints did not require complex modeling or coding, the evaluation participants mostly saw cyclic development in general as relevant (4.13) and also positively rated the benefits of a cyclic MDWE process (4.00). Moreover, all participants could identify the advantages of code and model synchronization (4.33). All in all, the perception of participants towards our approach was very positive and we consider it as a

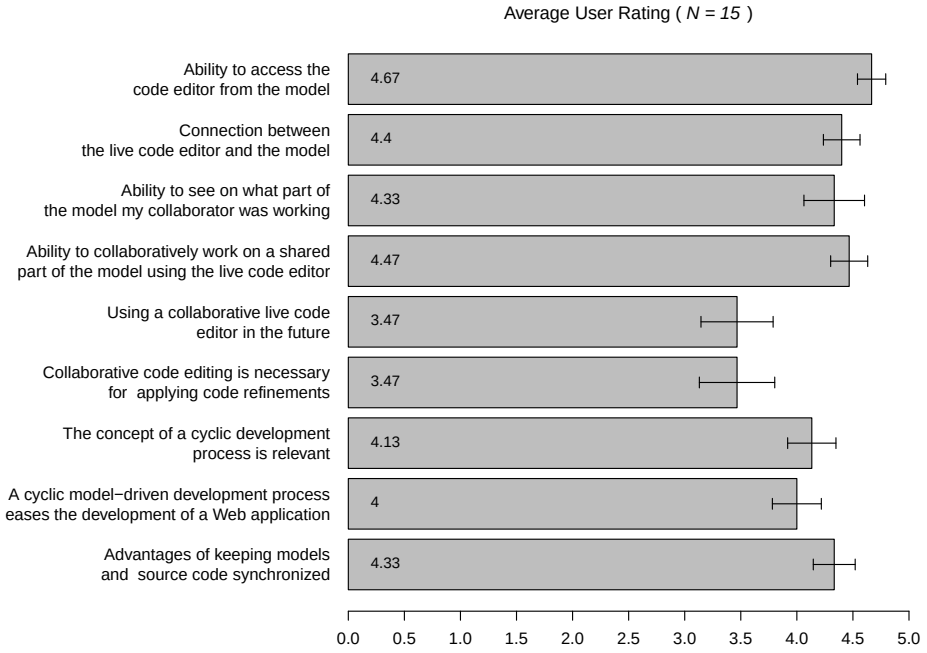


Figure 6.11: Results of the user evaluation

successful step towards evaluating our prototype using a more complex application in a real-world development scenario.

In order to evaluate the influence of trace generation on performance and space requirements over time, we measured the code generation time during the complete evaluation and analyzed one frontend component Git repository of the final resulting application. Here, we consider that even though it is isolated, due to our template-based approach, this case gives a good approximation about the behavior of our prototype in various scenarios. Since we used one repository for performing all sessions, we were able to consider data from 490 commits. The exemplary frontend component had a final total size of 634 KB, which contained 9 KB of generated and refined code. The static JavaScript libraries, images and other files added up to 173 KB. The final trace information occupied 42 KB. This leaves 410 KB of Git history, of which the trace history occupied 338 KB. All in all, the trace information, even though it occupies considerably more space than the code output, does not negatively impact the space requirements in a usual Web development setting. Especially, since it only scales with source code changes, that usually do not make up the larger part of an Web application in comparison to media assets and data. Moreover, the time to store, commit, push and retrieve code with trace information – considering our evaluation scenario data and participant's subjective opinion – does not introduce observable delays in the development process.

6.5 Summary

Here, we presented results and observations concerning CAE's realization, completing the design science approach described in the previous three chapters. We linked the CAE design into the ATLAS community conceptual framework and summarized the architectural and technical principles of CAE. We have shown how the framework builds upon SyncMeta and Yjs and how it supports various DevOps and continuous integration concepts in order to realize a modern, agile cyclic development approach with support for state-of-the-art Web engineering techniques.

CAE was used in our department for Bachelor practical courses for two consecutive years. It has been also presented at national and international conferences, such as Modellierung 2016 [dND+16] and the 23rd International Conference on Collaboration and Technology [dNKJ17], as well as in the Learning Layers EU project consortium meetings. CAE is part of the final Learning Layers deliverable [KKd+16] and an integrating part of the unified and scalable infrastructure developed by the project in order to support informal learning.

In contrast to collaborative coding approaches such as [DR93, GLM, DASH09, HL10] that introduce the concept for supporting pair programming, outsourcing, etc. and enabling the NRT collaboration on code level (also together with industry and community-lead projects such as EtherPad and SketchPad, Ace, CodeMirror, Ymacs, Kodingen, CodeBunk, Cloud 9 and many others continuously emerging on the Web), we lay the ground with CAE for a new approach for system (co)design using a Web-based editor for componentized, community-oriented and NRT collaborative applications. We consider CAE to be in a rather incipient phase still, as it is mostly oriented towards building Web widgets and therefore it only supports one community metamodel. It is more oriented towards the realization of customizable, scalable and community feasible frontend components. Software engineering literature covering collaborative modeling and NRT model synchronization [BPE+10, HD08, CSC+09], focuses on remote software developers specifying architecture, developing code in (large) distributed teams, the conflict resolution and awareness mechanisms, relevant user interfaces, etc. In contrast, our approach is community-oriented, focuses on developer support only as an intermediary step towards scaffolding all members in CoPs to get involved into the development process and it tries to propose a new way of collaborative conceptual modeling, as well as a novel and simple way to integrate NRT shared editing features into developed applications.

If we knew what it was we were doing, it would not be called research, would it?

Albert Einstein (1879 - 1955)

Chapter 7

Case Studies

Summary This chapter presents the case studies realized using the main software artifacts of the dissertation, following the methodological principles of the Community Application Editor. They represent Web applications generated or developed for professional CoPs, using our community-oriented, NRT collaboration enabling approach. We present a NRT collaborative storytelling tool created based on SyncMeta and two widgetized prototypes developed for professional communities working with multimedia artifacts, such as videos and 3D objects. We bring evidence on CAE's usability and our NRT collaboration concepts for online CoPs and describe – also from a design science perspective – the resulting Web CIS.

Contribution RQ1, 2, 3. *Keywords:* User Studies; Community Information System; Online CoPs; Non-linear Digital Storytelling; Web-based 3D Objects; Collaborative Video Annotation; The results presented here have been published in [NK14, NRK+14, NTK15, dNKJ17]. The next sections contain partial information and content extracted from these publications.

7.1 Multimedia Community Information Systems

7.1.1 Problem Identification and Solution Objectives

In the context of online professional CoPs from cultural heritage or construction industry, a great deal of research (cf. Chapter 2) studied the usage of multimedia artifacts such as images and videos using Web and mobile systems ([Kla10b], CAELUS [Kov14], YouTell [Han14, Cao12, CHK+10], SeViAnno [RCLK10, CRJ+10], etc.).

The development of SeViAnno 2.0 is a typical case of application reengineering and widgetizing scenario, trying to cope with the needs of such CoPs and to allow for more

flexible solutions that can address all the requirements exposed in the beginning of this research (collaborative, NRT, scalable, customizable, standardized, etc.). The major problem here – as we have encountered when working with real professional communities in EU projects context – is to cope with changing requirements within a community and even more, to be able to deploy resulting tools in multiple communities with minimal customizations and developer efforts as possible.

This is why several interfaces for the semantic video annotation application SeViAnno have been developed, until it was reengineered into the widget-based architecture of SeViAnno 2.0 presented in Chapter 2. In our case, we extended the support for collaborative annotations and NRT collaboration for specialized communities, also beyond videos. An example is the identification, together with expert members from the cultural heritage domain, of the opportunity offered by digital representations of 3D objects, that can be displayed and handled directly in Web browsers. Furthermore, another trending aspect is the personalization of video and multimedia content, based on metadata and community users preferences. Similar the IMS LD case, the main requirement is to enable NRT shared editing using the multimedia objects, for the specific professional CoPs. Concerning SeViAnno 2.0, the main requirement consists in the redesign of its architecture from a cloud-based backend solution to a modular, microservice-oriented one, using our CAE approach. The existing microservice components form a complex landscape of various functionality, that can enable features such as video adaptation and personalization in order to scaffold informal learning for certain user groups or CoPs. The resulting annotation microservices can be used in various community settings and with various multimedia objects (videos, images, 3D objects). A further use case resulting from the CIS setting is the collaborative browsing of 3D objects in the Web browser, making use of the annotation microservices for collaborative metadata creation around these objects during their exploration.

7.1.2 Artifact Design and Development

Video Adaptation and Personalization: Vaptor

The system[1] functions illustrating the video upload, tagging and personalization process used to support informal learning at workplace, are presented in Fig. 7.2. Here, members of professional communities act as video uploaders (or producers), video annotators or video consumers. Typically, the video producers and the annotators are more experienced community members, that want to share information with others. The annotation process is collaborative, in order to gather and use the non-obvious, domain-specific information of its members.

The figure also presents two targeted possible phases. We distinguish between a “preparation phase”, using existing tools (such as SeViAnno 2.0) and the “adaptation and personalization

[1] This section contains content published in [KNSK17]

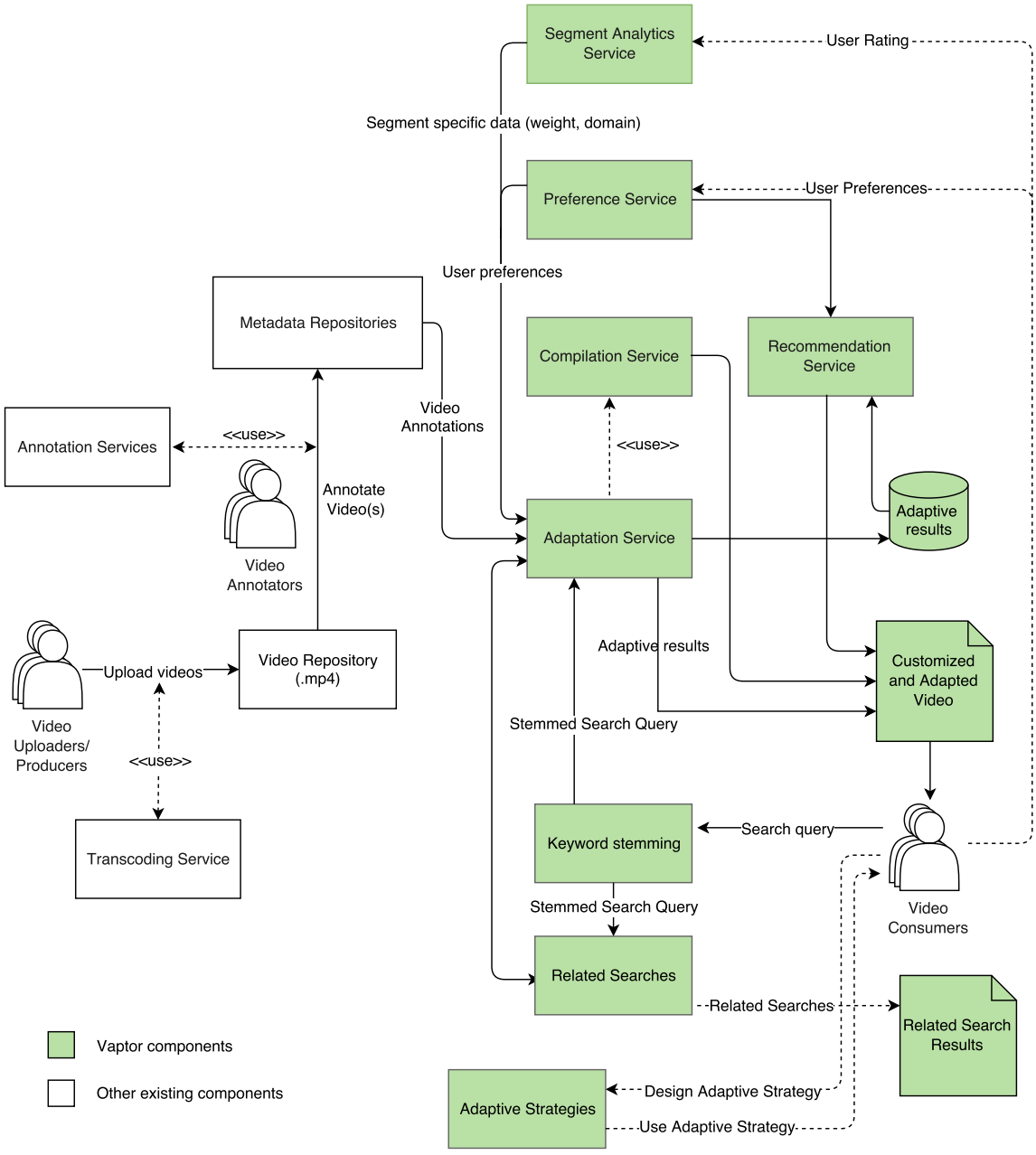


Figure 7.1: VAPTOR scenario

phase", which uses the existing user and video metadata. Next, we elaborate on these two steps.

Video uploaders upload the video into a shared repository, available to all community members. Before persisted in the repository, each video is transcoded by a corresponding microservice, in order to produce a common, well defined format for the multimedia objects used by the community. The resulting videos can then be annotated collaboratively with textual information (such as keywords) or predefined annotation types (for example Place, Object and Event tags as described for SeViAnno 2.0), by video annotators. The metadata is stored in a dedicated repository, which is accessible through an annotation microservice. With an existing pool of available annotated videos resulting from this initial phase – based on the available annotations –, the personalized and adapted video consumption takes place (c.f. highlighted green area on the right half of Fig. 7.2). The personalization is realized using the Vaptor microservice. Based on video predefined consumer profiles, upon keyword-based search queries, this delivers personalized video data and recommendations to video consumers. The recommendations may contain similar searchers and corresponding video compilations.

Each search query is first transformed by the "Keyword Stemming" component, which removes all the stop words from the query's text and forwards it to the "Adaptation Service". The "Adaptation" component receives from the "Annotation" microservices all the related metadata. The preferences for an authenticated user are retrieved from the "Preference" microservice. The "Adaptation" component also retrieves specific video segment metadata (such as domain, weight) for each annotation from the "Segment Analytics" microservice.

Based on the retrieved data, the "Adaptation" microservice compiles the results for each video consumer. These are directly displayed in the Web browser and also transmitted to the "Compilation" microservice, which transforms the received video segments into a new video and stores it for further usage. The "Keyword Stemming" component also stores the stemmed query among all requests for th related query functionality.

Changes to the user preferences are performed by the "Preference" microservice. As already mentioned, this component is used by the recommender system. The microservice also stores adaptive strategies defined for each user, that can be chosen during each performed search query. Furthermore, users are able to rate video segments through the "Segment Analytics" microservice, which assigns and updates the weights to the video segments based on preference data such as video duration, geolocation, etc.

The user profile contains variables influencing the video adaptation. Based on empirical data gathered from an early user evaluation, the profile contains for each registered user information regarding the language, location, preferred video duration, domain of interest and the level of expertise in a certain domain. The latter variable affects the rating and recommendation modules (the recommendations from two community members with different levels of expertise have different importance weights).

The user model of Vaptor is based on the following attributes:

- Search refinement parameters: language and location of the video
- Duration of the resulting video: limit in seconds (avoids very long videos)
- Semantic preferences: domain of interest (topic) and user's level of expertise (for the respective topic); topics are community-defined
- Relevance hierarchy: calculated based on the similarity between search query and metadata (like tags, description, title) of the annotation

The adaptation in Vaptor is based on a sequence of pipes and filters. Users can design their own, personalized adaptive strategy, choosing various filters and applying them in their desired sequence. An example of a possible sequence is as follows: after the search is performed and metadata results are collected from the "Annotation" microservice, the annotations are first ordered based on the relevance of the query and their semantics. Then, they are filtered based on the language. The remaining results are taken as an input by duration trimming, which ensures the duration of the resulting playlist does not exceed the length of user's preferred duration. Finally, annotations are ordered either by location or segment weight.

An adaptive strategy is mainly built up from filters that are applied on the data and their order. Once defined, strategies can be used by all registered users while performing a search. This feature provides users with a greater control over the obtained results. The following video adaptation filters are predefined, in accordance to the user preferences:

- Relevance ordering: calculated based on the text similarity between search query and metadata of the annotation (for example tags, description, title)
- Language filtering: the preferred language of the user is compared with the language of the video, in order to filter out non-matching objects
- Duration trimming: the playlist is composed taking user's preferred duration into consideration for the final result
- Location ordering: video segments are shown based on the user's location
- Segment weight ordering: the playlist is ordered based on the weight of the video segment, which depends on its rating

Architecture and Realization

Vaptor is generated using CAE and thus implemented as a Web application composed from widgets and microservices. Some of Vaptor's components (cf. Fig. 7.2) are existing microservices, part of the Learning Layers project testbed: Cloud Video Transcoder (ClViTra) and Semantic Video Annotation (SeViAnno 2.0) [KKd+16] microservices. The communication

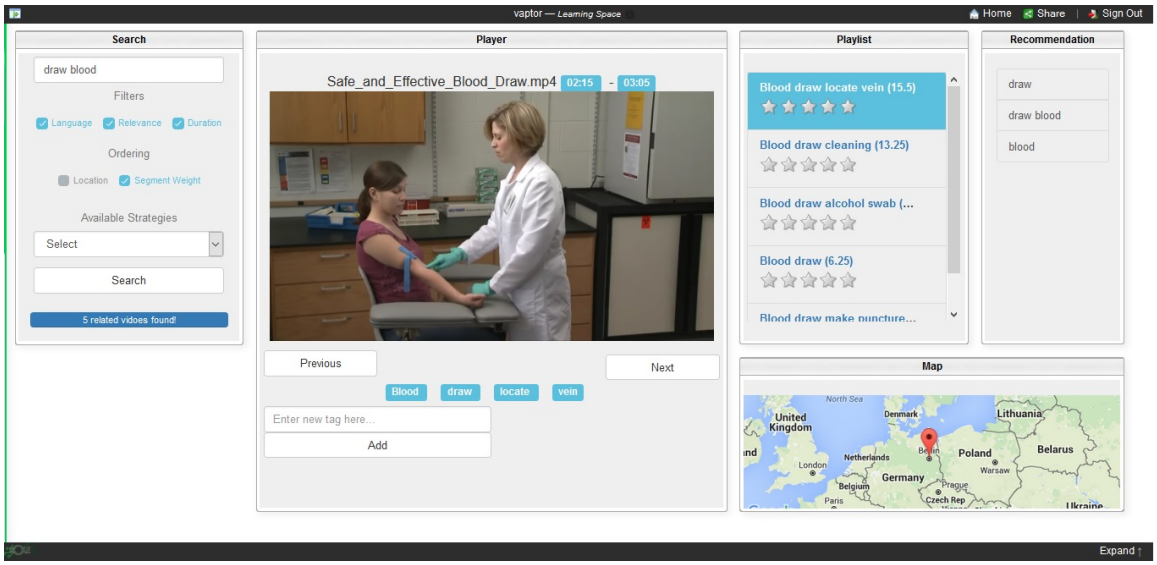


Figure 7.2: VAPTOR screenshot

with the SeViAnno 2.0 reengineered backend components and ClViTra is done using their respective RESTful APIs.

Vaptor's microservices are designed to perform dedicated tasks and support the CAE's architecture principles. The aim is the fast deployment, and update of microservices. The Vaptor microservices are: "Adapter Service", "Video Compiler Service", "User Preference Service", "Recommendation Service", "Orchestration Service", and "Segment Analytics Service".

The user interface of Vaptor (cf. Fig. 7.2) is based on twelve ROLE SDK widgets. They use both the IWC generated by CAE, as well as the RESTful interfaces of the las2peer services. Widgets under Vaptor are placed in three different areas, called activities: 'Video activity', 'User preference activity' and 'Alternative adaptation activity'.

Anatomy 2.0: A Lightweight Collaborative Annotation Approach for 3D Objects

In order to realize a CIS beyond Web videos and video annotation, we concentrated our efforts in achieving a usable Web platform where learners can interact collaboratively on the Web with 3D objects[2], in NRT. The Anatomy 2.0 prototype was therefore designed to be used as a learning platform for medical students, as a new paradigm of interaction with digital artifacts (anatomical scanned objects). This decision was also supported by the usage of our ROLE SDK testbed, that offers the implementation of a PLE and therefore is suitable in such learning scenarios.

[2]This section contains content published in [NTK15]

Therefore, for a specific course (for example taken from medicine curriculum), users can collaborate on and explore 3D objects in a shared space (PLE environment). By entering the corresponding learning environment, learners can browse 3D objects directly in the Web browser, in two modes: a personal mode, where they explore the objects independently and a collaborative mode, where an object of focus is shared to all learners. With this mode active, all interactions upon the object are synchronized in NRT. The available actions on the 3D object allow zooming and rotation to arbitrary camera views. The synchronization function allows the tutor to share a specific camera view to all the participants of the virtual space. They are also allowed to collaboratively annotate the currently explored objects. The space simulates the real world experience both in terms of freedom of interaction with the 3D model and awareness of the participants among each other, a stimulus regarded essential in conventional classroom learning.

For this purpose, we have used our CAE approach for realizing several widgets. The annotation interface is presented in Fig. 7.3. Each space is dedicated for the collaborative study of anatomical objects and corresponds to a course. The managing of course registration, curricula, etc., is realized through a separate Website, that offers functions for managing the courses, the spaces, uploading the 3D objects and assigning them to specific courses. Fig. 7.3 shows a space (in the PLE) used for an Anatomy course, according to the above described scenario.

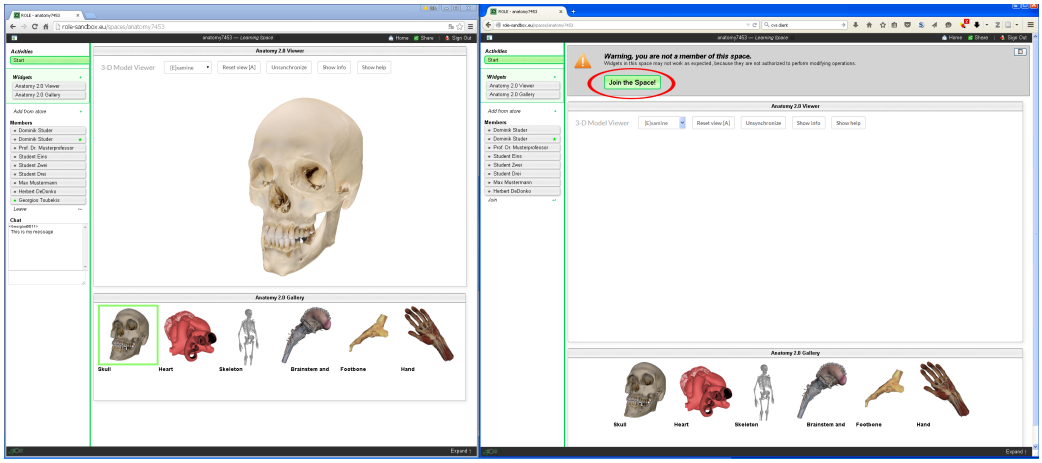


Figure 7.3: Shared space for exploring 3D anatomical models

As it can be observed, the space is composed from two widgets, one being a repository of existing 3D Objects to be explored and a visualization widget, that displays one 3D object at a given time. The 3D Object Viewer widget is built using the X3DOM library and features various examination modes of the objects (examine, walk, fly, etc.). Learners or tutors can synchronize (receive and visualize the updates of other users based on the same object, powered by Yjs communication) or unsynchronize (ignore updates) objects. They can reset the view to its initial spacial position and camera view angle and show X3DOM information

about the current object, such as hardware rendering, number of nodes, number of points. All participants interacting within a space that has the synchronization mode enabled will have the same view on the same object. Each participant can interact freely with the object of focus.

During the annotation process, tutors (or learners) can use the SeViAnno 2.0 microservices and its defined annotation types in order to describe the 3D object. The annotations are performed at the meso level (at the 3D object level) in the PLE [CKK+15]. The community uses the annotations in order to exploit knowledge (annotations are used as a method of underlining relevant or interesting aspects of an object) and as reflection possibility for the case where a learner explores others' annotations. We use simple text annotations containing a title, author and a description. Furthermore, our annotation approach also provides a range of basic meso level annotations that can be used to disambiguate meaning [CRJ+10]. From the expressiveness point of view, we achieve a highly customizable tagging structure that can be used for rapid prototyping and guarantees a good and expressive annotation experience for learners and tutors involved in the process.

Architecture and Realization

Our technical approach focuses on achieving a collaborative environment using CAE that can easily scale with the number of learners. Therefore, we realize the architecture instance presented in Fig. 7.3.

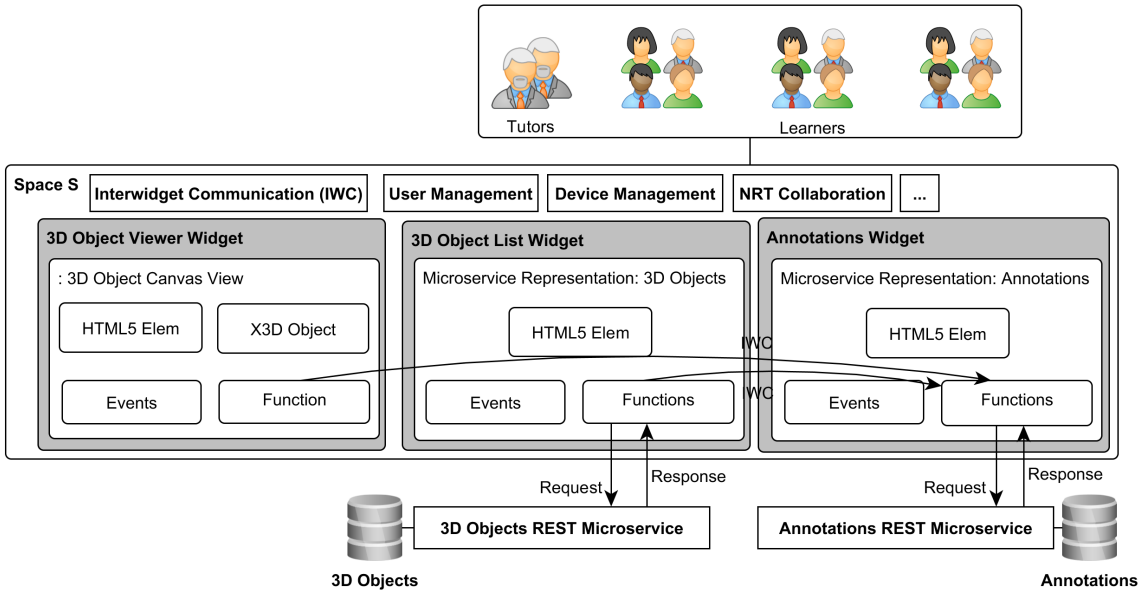


Figure 7.4: Architecture instance for rapid prototyping and lightweight annotation services

The lightweight approach proposes dedicated microservices for fulfilling clear functionality such as multimedia annotation (SeViAnno 2.0 components) or multimedia storage (for the 3D objects), some being reused from the SeViAnno 2.0 and Vaptor context. The

microservices are reflected on the interface level using widgets such as the 3D Object List seen in Fig. 7.3 or the Annotations Widget. The latter annotations widget is partially reused from the SeViAnno 2.0 available frontend components. A full reuse was not possible due to the different visual representation of the annotations (conic markers) available in the 3D object annotation use cases. Upon need, these frontend components can be reused together with other widgets (such as SeViAnno 2.0 or Vaptor widgets).

As depicted in Fig. 7.4, our environment uses two microservices: a microservice for the 3D objects and another one for the annotations. The 3D object microservice uses a MySQL database to store information about the 3D objects such as title, description, URL, learning environment address to which they belong, etc. The annotation microservice is reused from SeViAnno 2.0 and Vaptor and uses ArangoDB, a non-SQL hybrid (graph, document, relational) database written in JavaScript. The annotations are stored inside the service as a graph, using a JSON format. We use two main graph vertex types (collections), which have unique object identifiers. They represent objects and annotations and are stored with their full JSON representation. If an annotation is added to an object, a vertex is added to the graph and an edge is created, that links the annotation with its corresponding 3D object. The contextual data concerning the annotation and the respective object (annotation time, absolute coordinates of the 3D object, annotation position, rotation and translation information) is stored in the edge, also as a JSON representation.

Annotation types can be easily added upon learner's needs and rapidly reflected into the user interface. Since standardization for annotations is still an open issue, we propose to achieve interoperability by specialized conversion microservices, that can transform the unstructured JSON annotation representation into more structured standards such as MPEG-7, DublinCore, etc. This is how a transformation between an existing annotation and a dedicated standard (with the goal to include annotations as native labels in X3D) can be achieved.

Finally, the 3D Object Viewer does not have any connection to a microservice, but it listens for IWC messages from the other widget events, in order to get information regarding its state (such as object to display, annotations related to current view). The other widgets are displaying information retrieved via asynchronous RESTful calls from the microservices. The widgets are composed from events, functions and HTML5 elements. All NRT shared editing functionality (synchronization of 3D Objects, rotations, translations) is managed using the Yjs library.

7.1.3 Evaluation

Vaptor Evaluation

Methodology In order to validate the user experience with Vaptor and its relevance for informal learning, we have performed a user evaluation in a simulated community environment. The evaluation scenario was divided into three tasks. The first two tasks assessed the

contribution of Vaptor to informal learning, whereas the last task evaluated its usability and individual features.

In the first two tasks participants were asked to search for certain terms in a predefined collection of videos, in an effort to learn certain notions on a given task. In order to assess the difference compared with existing non-adaptive applications, one task was performed using Vaptor (with predefined adapted videos) and another using the SeViAnno 2.0 application. After participants viewed the video results, they were asked a set of questions, to evaluate their learning process. In the third task, they were formally introduced to Vaptor and given a set of keywords, they were allowed to freely search and explore the application. Finally, participants were requested to fill out a questionnaire, comprising of questions about the Web application, its features and its usefulness.

The first two tasks contained videos with information about microchips and videos from the healthcare domain. Before each of the first two tasks, participants were asked to rate their own knowledge about the topics. Afterwards, they were asked questions about certain domain details and left to search for videos containing the right answer. Participants were not knowledgeable in the given topics, therefore simulating a novice community member.

Results Altogether, we had 20 participants that used Vaptor in individual sessions. 14 participants had a Computer Science background, whereas 6 were students in other study areas. Concerning the process of learning about a microchip's construction, the average percentage of correct answers when participants used Vaptor was 60%, whereas while using SeViAnno 2.0 it was 43%. Making the same comparison for the second task (learning about a cardiopulmonary resuscitation procedure), participants scored on average 88% using Vaptor, whereas using SeViAnno the average score was 80%. These scores are based on the number of correct answers given to the questions for each topic. Even though the number of participants was rather limited, the findings suggest that Vaptor keeps the users focused on a specific topic. This can be interpreted as an indicator towards informal learning.

We also analysed the time spent by participants for going over the two topics. For the microchip task, they spent, on average, 05:03 minutes using Vaptor and 08:33 minutes using SeViAnno 2.0. The time for the resuscitation practices, was 04:32 minutes on average using Vaptor, and 07:12 minutes using SeViAnno 2.0. Each system used the same video and annotation repository for their tasks. However, since Vaptor displays only relevant data, the created playlist was slightly different for the same topic. The playlist was adapted to the user's given profile and search query in Vaptor. The drawn conclusions after this experiment are that Vaptor makes learning more efficient by reducing the time required for searching non obvious information. Given that the user generated profile matches the video metadata, it behaves better than usual video-based systems.

We also evaluated the user definition and usage of adaptive strategies. The use of adaptive strategies was favored by 60% of the participants, whereas the definition of adaptive strategies was favored by approx. 50%.

Anatomy 2.0 Evaluation

Anatomy 2.0 was evaluated within a blended learning project providing access to 3D anatomical models as described above.

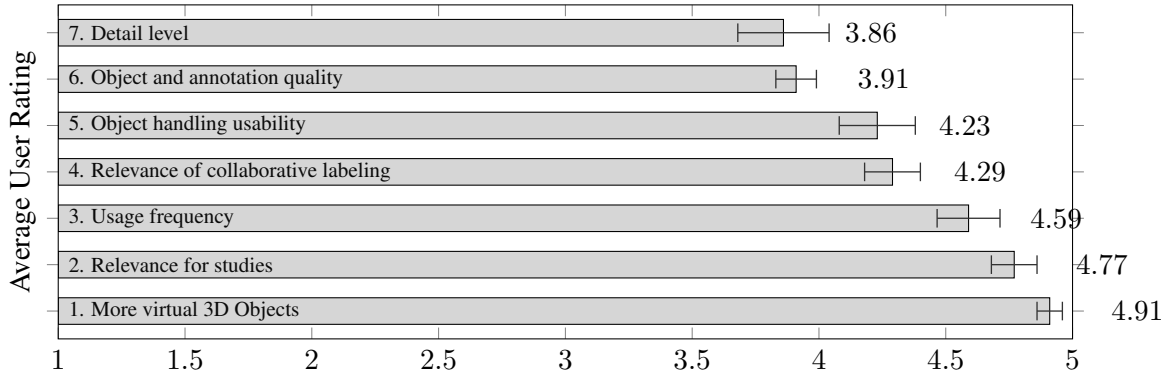


Figure 7.5: Evaluation results

In order to test the feasibility for learning and the usability of our approach, we performed two study sessions with students from the medicine faculties of two different universities. 24 participants from Aachen and Maastricht used a PLE to visualize the 3D objects and annotate the non-obvious information of the anatomical models. First, a tutor provided information on certain 3D objects by using the synchronized view. Then, students were allowed to visualize and annotate independently or collaboratively the objects.

This setting was created in order to test the performance of the annotations service and how the 3D Objects are perceived in the browser by the users.

We measured the acceptance of the system using questionnaires based on a five points Likert scale (cf. Fig. 7.5). The overall acceptance towards the current version of our prototype was very good and the usage of the models where characterized as helpful. However, the relevance to the medical subject requirements and learning goals still remained unclear. Based on the available system, the evaluation should be extended with more scenarios.

7.1.4 Communication: Discussion and Lessons Learned

The SeViAnno 2.0 application, its componentization and customization needs were presented and discussed at learning and multimedia related conference events, CBMI 2014 [NK14] and ICALT 2014 [NRK+14]. This was used, together with the Vaptor prototype in the context of the 7th Framework Programme large-scale integrated project Learning Layers (grant no: 318209). Vaptor was presented at the First International Symposium for Emerging Technologies for Education workshop at ICWL 2016 [KNSK17]. Finally, Anatomy 2.0 was presented at the ICWL 2015 conference [NTK15].

All in all, we presented a lightweight approach based on our Community Application Editor for the collaborative exploration of 3D objects on the Web, using annotations and NRT mechanisms in learning practices using adaptive and personalized videos. The resulting prototypes are accessible as open source and can be combined in a PLE, achieving an interplay between 3D technologies, annotations and NRT collaboration in order to facilitate CoPs learning practices on the Web. Vaptor's evaluation shows the capability to support and contribute to online learning for professional CoPs. Furthermore, the evaluation performed using anatomical artifacts demonstrates the relevance of these technologies for education and training purpose.

As lessons learned, since the NRT collaboration and rapid prototyping of such components can be easily achieved, we observe the need to support more annotation techniques and explore the vocabulary formed by the communities in a structured evaluation process. Our CAE-based provided realization is still quite general in terms of semantics and expressiveness. A domain-specific annotation process, by also keeping the lightweight characteristic, could probably be integrated even closer into the practice of communities. Further directions worth exploring are the distribution of different widgets on multiple devices and how can a seamless experience influence learning in such collaborative settings. Finally, community monitoring and success measuring [DM03] methods can be integrated in the resulting CIS, in order to extend the evaluation of the systems for learning communities and to provide self-reflection mechanisms [Kla13, RKKN15] for tutors and learners.

7.2 NRT Digital Storytelling for Learning Communities

7.2.1 Problem Identification and Solution Objective

In a social setting, the usage of digital media and Web 2.0 have a great potential to facilitate the creation and distribution of digital stories, as a mean to transfer information. Digital storytelling puts the individual back into perspective and can be used in professional CoPs as a tool for learning and knowledge creation. Collaborative storytelling helps to achieve a common understanding within groups, using narration in order to transform implicit knowledge into explicit one [LKB11]. Using multimedia objects in digital storytelling bears the potential to improve conveying story content and offer a richer experience, beyond simple text and conventional Web content [SBL13]. Moreover, hidden, we already state that non obvious information surrounding multimedia artifacts can be enriched using annotations, that highlight the most important, interesting facts. Due to these advantages, in the context of the Learning Layers project, we decided to investigate the learning effects in professional CoPs obtained by combining the two techniques (meaning storytelling and annotation).

One popular practice in (interactive) storytelling creation is delivering non-linear stories, as story consumers can influence the direction of the narrative with their actions and creators have access to a richer mean to express their narration. Allowing the creation of non-linear

stories provides more degrees of freedom for both the authors and consumers. Especially in many informal learning scenarios –which involve people working together–, enabling the collaborative creation of stories in NRT facilitate a participation culture, knowledge negotiation and thereby finally lead to more productive results.

Recently, transforming real-world artifacts into a digital format has become easy and cost effective, enabling access through the Web to a huge pool of potential users. An example is given by 3D digital representations of real-world objects that are otherwise not reachable for the majority of learners, due to their perishable or expensive nature, as presented in the previous case study.

We have therefore identified the need for a Web-based digital storytelling approach, which can be used to create and share multimedia-based non-linear stories around 3D objects in a collaborative manner. The focus of these stories is placed on informal learning, which is meant to be supported through the mixture between collaboration in the creation process, annotation of 3D object views and camera perspectives with various multimedia such as text, videos and images and novel, intuitive consumption the resulting non-linear stories.

7.2.2 Artifact Design and Development

We have designed and implemented a non-linear digital storytelling concept around 3D objects using SyncMeta and the Community Application Editor. In our approach, we use the *Movement Oriented Design* (MOD) [Sha07] paradigm to provide a structural guideline for authoring the stories. One of the goals is to model the MOD paradigm using a SyncMeta metamodel and instantiate this metamodel using widgets and microservices (as specified in the Community Application Editor concept), into a digital storytelling prototype.

For describing our solution, we consider two scenarios, “story consumption” and “story editing”, that showcase the requirements and exemplify the contributions of the collaborative digital storytelling artifact.

Consuming Stories. To consume a story, a member of a community interested in guitar playing lessons, chooses a 3D guitar object already captured by other community members and selects to visualize a story around this object, from a list with available ones. The story is presented next to the 3D object, such that community members can browse freely the 3D object or start their “journey” through the story. Stories are split into units, each unit corresponding to a story page and to a specific camera view angle on the 3D object. The available stories are non-linear. After each story page, there is either a single transition to the next page, or a decision between multiple possible steps, from which story consumers select one. Hence, each learner can experience different story paths while navigating through the guitar playing story. While clicking through the story pages, the camera automatically moves around the 3D object, focusing on different parts of it according to the story content. Besides these automatic camera movements, story consumers are free to also move the camera manually, in order to have a closer look at certain aspects of interest. The connection

between the current point in the story and the 3D object is marked with metadata tags, appearing as linked on the object. By clicking the annotations, story consumers get detailed information on specific features. The detail page contains text, images, links to external Web pages and videos. When a consumer reaches one of the possibly multiple endings of the story, an alternative path can be chosen by returning to a fork encountered in previous story pages.

Story Editing - Adding Content. During the story consumption, the story consumer encounters an explanation of one of the guitar's features, that can be improved with own knowledge. Using the story browser, the consumer opens a story editor instance and locates the story unit considered for improvement. To highlight an important feature of the 3D object that has not been properly described, he places a tag corresponding to a camera view on the digital guitar and fills in a short textual description and a video link found online about the respective feature. Now in an editor role, if the consumer wants to input a very detailed explanation, he can also create a small substory inside the existing story from scratch, and connect it to the overall plot. During this whole process, other community members can edit or browse the guitar object simultaneously in NRT.

Story Editing - Creating a New Story. A community member is an expert on a certain type of guitars and has access to a 3D model of it. In the story viewer, he chooses to create a new story, which directs him to the story editor. First, the story author and media producer connects the 3D model with the newly created story. Then, he starts to scaffold the story by first modeling the coarse structure in form of a MOD tree graph, without adding media content yet. He leaves text notes about pending clarifications in the various story section nodes. Noticing some lack of knowledge about a certain aspect of the guitar object, the community member asks an expert for help. The expert joins the story editor and after a brief introduction (either via the chat provided by the editor itself, third party communication tools, or face-to-face communication) he starts building his substory. They then start adding their multimedia content to the story graph. The story creator places tags on certain places on the 3D object to highlight those, and connects the tags with appropriate story sections. Together with the expert, they also define camera angles that best highlight relevant aspects of the object they are currently exploring. When the story is in a presentation-ready state (if the story is well-structured and semantic rules are not being violated in the story's model), the editor decides to export it, such that the object and its attached stories can be consumed by other community members.

Movement Oriented Design

Many good examples for educational storytelling include the listener expecting a story and being told lessons along the way. With MOD [Sha07], Nalin Sharda presents an approach that seeks a best trade-off between the advantages of storytelling and the rate of information presented to the listeners. MOD is a storytelling paradigm that tries to assist storytellers in wrapping up subject matter into compelling stories. A story is seen as a collection of *Story*

Units, where each *Story Unit* has three components called *Begin*, *Middle* and *End* (BME). The *Begin* of a *Story Unit* is meant to "hook" the consumer. The *Middle* carries the main information and the *End* terminates the story or links to the next story unit.

In MOD, the final story is one big *Story Unit*, each of whose BME components (also referred to as *Movements*) can be recursively subdivided into additional *Begin*, *Middle* and *End*, until the authors consider that no further decomposition is necessary. Sharda reduces the nature of a story to the arc of suspense in a concluding *Story Unit*. By enforcing the BME-scheme, the story creators are nudged into considering a basic narrative structure, having no upper limit on dramatic intensity.

Sharda emphasizes the applicability of MOD for non-linear storytelling, by instantiating a *Movement* with multiple *Story Units*, and/or by linking *Movements* non-linearly. An example of a story with several story units is presented in Fig. 7.7. A consequence of a non-linear interpretation of MOD is, that there is no longer a predefined sequence in which the story should progress. If for example the story has two *Middles* and two *Ends*, both *Ends* would have to be available to the consumer, no matter which *Middle* was chosen.

An example of a non-linear implementation of MOD is given in *Media Integrated Storytelling* (MIST) [SKSJ06]. Here the authors solve the problem with the ambiguous affiliation of multiple *BME* components by not allowing the recursive split of a *BME* component directly. Instead, if for example a *Begin* has to be further split into a *Begin*, *Middle* and *End*, those have to be combined under a new *Story Unit*, which is then associated with the initial *Begin*. The problem of unambiguous story paths is solved by forcing the author to define all transitions manually. The connections have to be drawn in a way, that each possible path through the story would result in a correct linear MOD story.

A Metamodel for Non-Linear MOD Storytelling around 3D Objects

We define a metamodel that specifies non-linear stories for 3D objects and connections between these 3D objects and media, tags and 3D properties such as position, camera angle, perspective, etc. Due to its suitability for Web 2.0 collaborative modeling, we choose SyncMeta to implement the MOD principles. Furthermore, we choose to implement non-linearity using the non-linear interpretation of MOD from MIST. The conceptual metamodel can be seen in Fig. 7.6.

The central entity of our metamodel is the Story Unit. It has a Title, as well as a Problem and a Solution. A Story Unit has to have exactly one Begin, Middle and End, which derive from the class Movement. All Movements have a Title and the option for the author to leave a Note. As in [SKSJ06], the recursive splitting of Movements into further Begins, Middles and Ends is achieved by associating another Story Unit with the Movement, which is then split up. The Media Objects constitute the leaf nodes of the tree structure resulting from this setup. They contain the actual media content presented to the consumer. Media Object itself is an abstract entity with a title, a caption and again a note, from which the

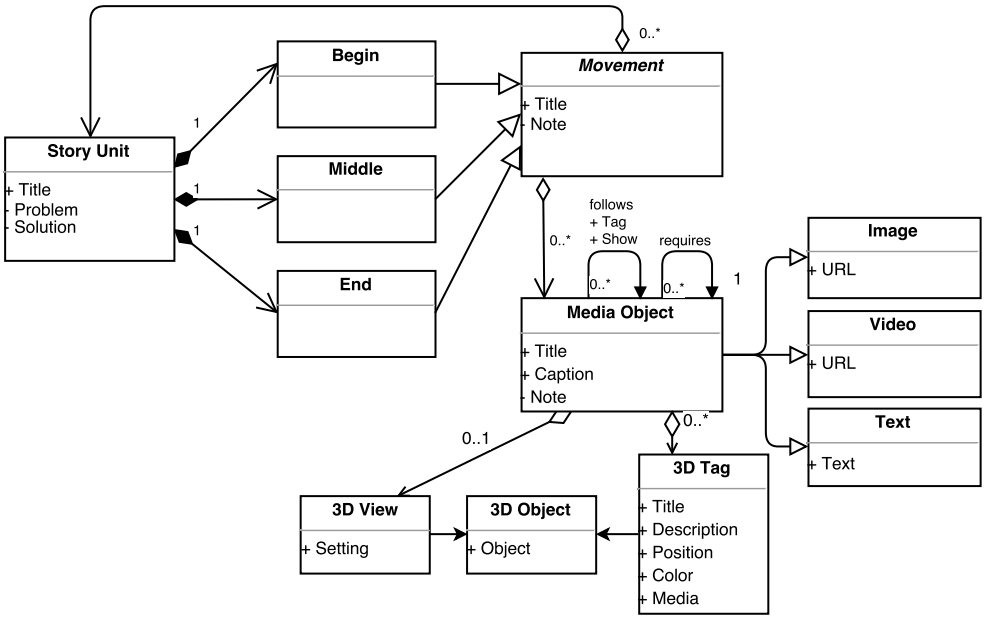


Figure 7.6: The YouTell 2.0 conceptual metamodel

concrete entities Image, Video and Text derive. They constitute the media nodes, that the story creator utilizes. Images and Videos have a URL attribute as a reference to the actual media file, while the Text has the equally named attribute text for the content. For dictating the story's flow, there is the *follows* relation between Media Objects. Each Media Object can have an arbitrary number of successors. These *follows* relations can also have a Name and a Tag, whose clicking (also) causes the transition. Each Media Object can be associated with an arbitrary number of 3D Tags. A 3D Tag has a title, a description, a color and a reference to a remote media file as attribute media. Its *meta* attribute stores the tag's location and orientation. Each Media Object can also have at most one 3D View, whose only attribute is the camera setting called *meta*.

From a conceptual point of view, it is also necessary to be able to associate the 3D Tags and 3D Views with the 3D Object they belong to. Additionaly to the MIST layout, we introduce *Requirements* relationships (called *Requires* in the metamodel). Their purpose is to enable the story author to save redundancy in the story graphs. If two nodes A and B are connected as $ARequiresB$, then while consuming the story, B will only be accessible after the viewer has already visited node A. This reduces the complexity of story setups, where two paths lead to the same or very similar states, which however result in a different progress of the story. However, they don't have an impact on the way to determine the validity of a story, as each story graph using requirement arrows, can be easily converted into one without such, while the story does not change from the consumer's perspective (cf. Fig. 7.7).

The metamodel presented is created using our SyncMeta domain-independent visual modeling approach and instantiated into story editors. During the story editing phase, many users

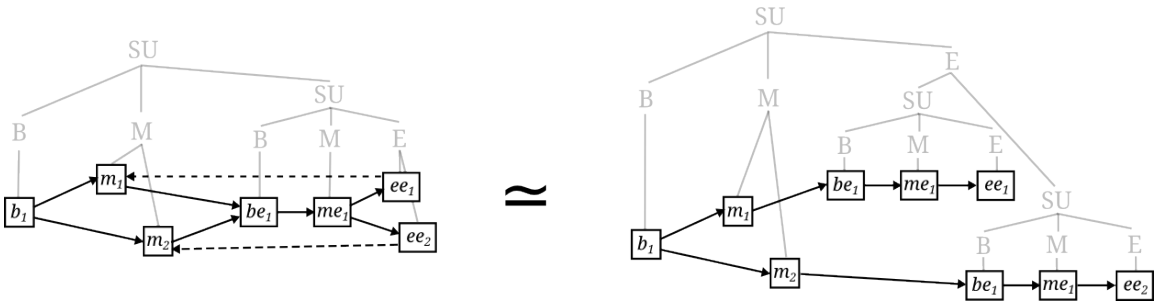


Figure 7.7: Equivalence of stories with and without requirement relations (dotted arrows)

(having various roles) can model stories collaboratively, in NRT in a shared space, while also interacting collaboratively with the 3D object. Once ready, the story can be viewed and consumed in separate views, as presented in the next section.

Artifact Realization

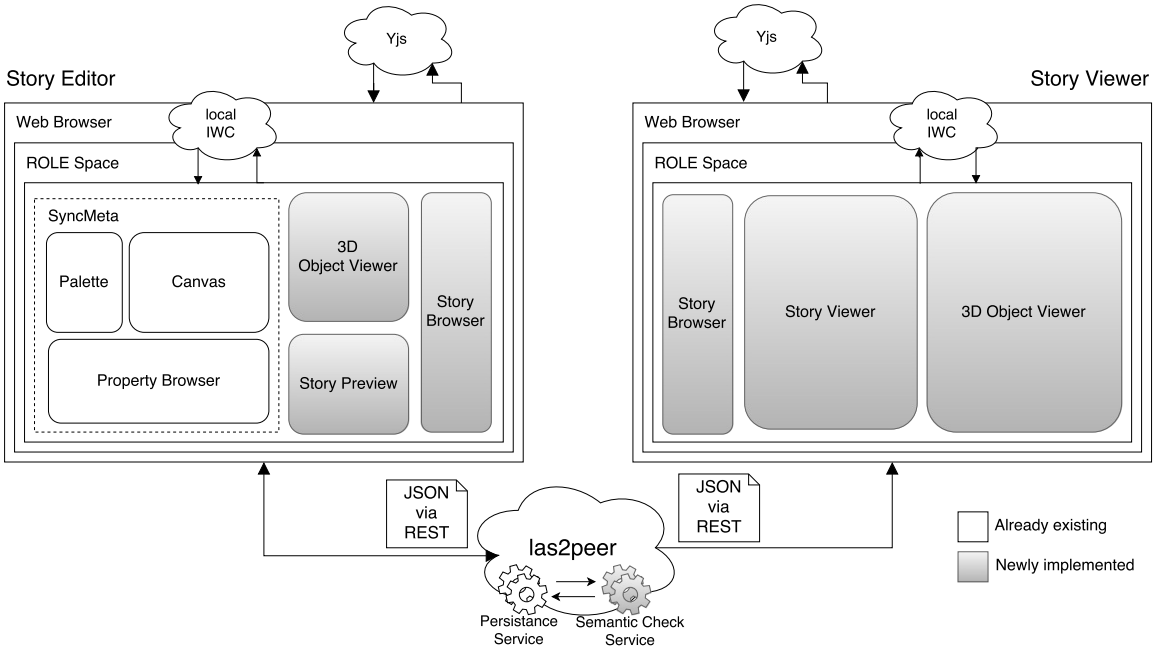


Figure 7.8: YouTell 2.0 architecture diagram

Our concept is implemented using a modular and flexible microservice-based architecture according to the CAE approach. The resulting artifact, YouTell 2.0, is a Web-based collaborative storytelling environment instantiated in the SyncMeta framework.

Fig. 7.9 presents a guitar 3D object and a story created around it, teaching how to play the instrument. Users can rotate and interact with the guitar and the 3D annotations appearing on the object's surface. In the Story Preview widget, they can visualize the story pages and navigate through the story, by clicking the next button or the previous link. Here, text, videos or images can be shown depending on the content of a story page. The figure presents the main Web widgets (for story creation, navigation, 3D object manipulation and viewer).

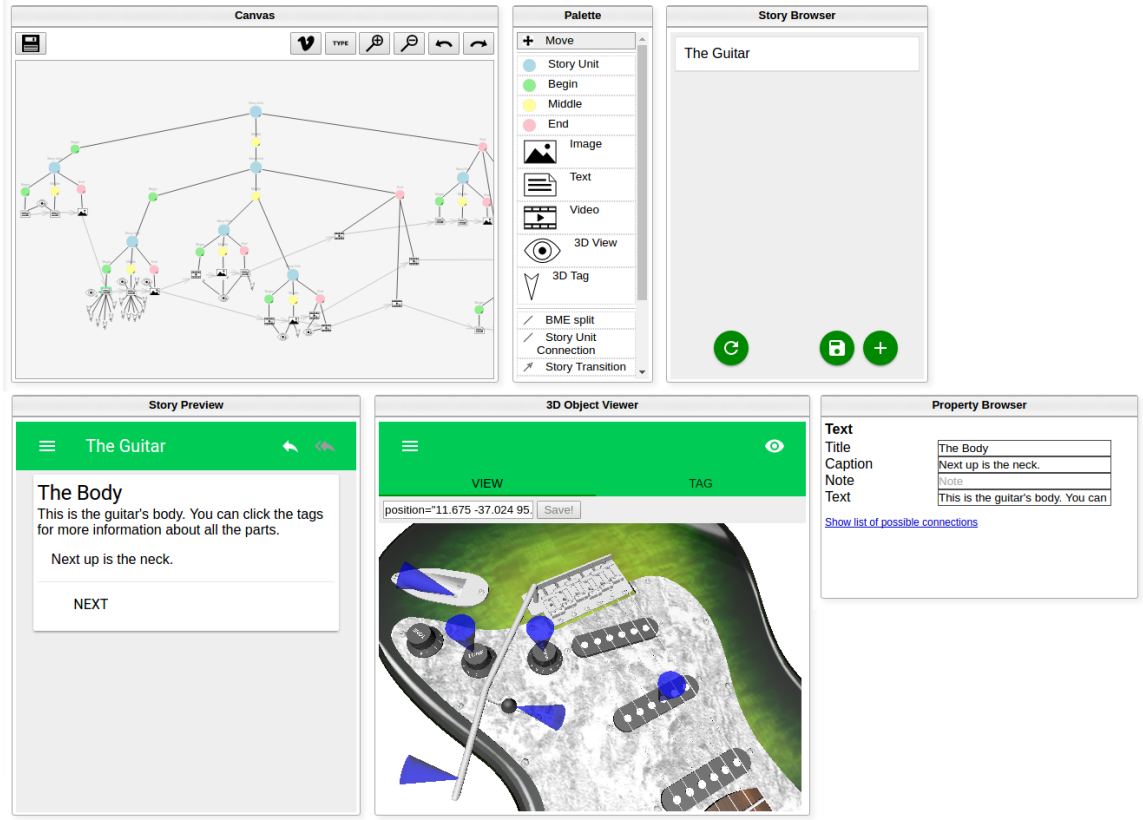


Figure 7.9: Screenshot of the story editor

Stories can be created collaboratively in NRT using the *Palette* and *Canvas* widgets, which reflect an instantiation of the metamodel. The *3D Object Viewer* widget is designed to contain the 3D-related features. Using it, users can adjust camera positions in the story editor, place and export tags and select the 3D object in the Story Viewer widget. By clicking on a certain tag, a window containing text, images and/or videos opens. A "story transition" tag triggers a switch of the story page. When switching the story page, the camera automatically moves to the 3D view associated with it (if available) and the corresponding tags are shown. Users can move the camera manually around the available 3D object.

The *Story Viewer* widget (named *Story Preview* in the editor widget) displays the individual pages of the story and provides the navigation. The editor content is updated when changes

in the story occur, in order to provide a live preview of the current state of the story. The *Story Browser* enables save and load functionality of stories. Backend RESTful microservices ensure the persistence of stories and perform semantic checks upon changes. When a semantic error occurs in a story (incompatibility with the metamodel), a detailed description is shown in the story's list entry. The system is available as a collection of open source projects in GitHub.

7.2.3 Evaluation

Methodology

We evaluated our storytelling approach, using the YouTell 2.0 prototype, twofold. First, we conducted an user evaluation to assess the general concepts with users without a specific profession or expertise in a particular field. The user evaluation has been conducted using the MobSOS community framework. As mentioned in Chapter 2, MobSOS provides rather complex and detailed analysis tools for communities, being also integrated into the ROLE SDK and its sandbox environment. Due to the investigative and empirical nature of this study, we have generated a questionnaire with open and likert scale questions using the MobSOS framework. We recruited our participant group from university students and staff, from different fields of study. We investigated the compatibility between storytelling and 3D objects, how well the non-linear interpretation of MOD from [SKSJ06] is suited for our scenarios and how the collaborative aspect of our solution is adopted and rated by the participants. Second, we evaluated our approach with a domain expert in the field of (teaching) medicine in order to assess the learning effects from a tutor perspective.

The user evaluation was conducted in group sessions of one hour with two participants each, resulting in a total number of 12 participants (6 sessions). After a short introduction to YouTell 2.0, the participants started to work on predefined tasks, provided on paper. The first task was to simply view a demo story prepared beforehand, in order to become familiar with the story consumption. In order to evaluate the intuitiveness of the *Story Viewer*, we did not provide a detailed user interface explanation on the task sheet. One of the story pages simply contained a "to do" message. When both participants of a session browsed through the story to one of the endings, we pointed them back to this page. Besides giving participants a hint on what they were about to edit, this also ensured that they used all functionalities of the prototype.

In the next task, participants had to use the story editor, in order to collaboratively replace the given story page with meaningful content. The evaluation instructions contained detailed information on how to achieve this. The task sheet also contained different roles for both participants, ensuring that they collaboratively create a short substory, integrated into the overall story. At the end of the session, the participants were asked to fill out a questionnaire with closed and open questions.

For the evaluation with the domain expert, we did not prepare a fixed set of tasks. The domain expert was provided with enough information to acquire experience with the tool autonomously during a longer usage period. After this time, he was asked to author a custom story from scratch. This allowed a least biased view on the system, giving us as much insight into its perception and room for improvement as possible.

As a guide, we provided a series of tutorial videos on how to use the editor and explained the concept of MOD. The expert was asked to fill in an interview-like questionnaire, containing also open questions. While this shared some questions with the first user evaluation survey, it contained specific questions around the concept of the story structure. It also inquired more details about the functionality of the editor. Additionally, we also conducted a face-to-face interview with the domain expert.

User Evaluation Results

One main focus regarding usability was the intuitiveness of the *Story Viewer*, implying the intuitiveness of the *Story Viewer Widget* and *3D Object Viewer Widget* in the *Viewer* configuration. This was rated well on average, with a score of 4.33 ($SD = 0.49$) Furthermore, the participant comments were very positive ("Very simple and intuitive"). For the *Editor* side, a participant stated that he understood and valued the usefulness of the tool, after working for cca. 20 minutes with it. As critique point, the participants commented that the transitions to next story pages from a given point were not obvious. An interesting observation was that many users only clicked the rightmost button in the first run, ending up at the same point of the story. This problem has been dealt with by highlighting the buttons, to better separate them from the background and each other. The *live preview* aspects of the widgets in the *Editor* space were rated with a score of 4 ($SD = 1.13$) . Although rather high, the feature requires slight implementation improvements due to its prototypical nature during the evaluation ("Widgets sometimes need to be reloaded", "Just sometimes I had to refresh the canvas by myself").

One of our goals was to evaluate the acceptance of non-linear stories for learning purposes. This was a complicated task, as the participants only had a short time to get familiar with the concept. They seemed to prefer the idea of non-linear stories over both linear ones, structured information without a story, and completely unstructured information. An average score of 3.58 ($SD = 1.13$) for "no structure" was the lowest of these options. Also very contentious was the question, whether the plot was helpful for keeping the participant interested. It was answered with average score of 3.4, but with $SD = 1.43$, this does not allow for a conclusion. Factors that might have influenced this result were the limited time of the sessions and of course the perception of the individual demo story we used. With an average rating of 4.5, the step back functionality was highly rated for revisiting information on a previous page and, with a very high score of 4.75 ($SD = 0.45$), for choosing an alternative story path.

The combination of 3D objects and storytelling was very well received, both conceptually

and in terms of usability. Another point was to investigate ideal use cases related to CoP learning processes of our system. We asked the participants, how much they would appreciate the non-linear storytelling for learning, in formal and informal scenarios. As expected, the results showed an increase in acceptance with decreasing formality with a lowest average score of 2.92 for university education, and a highest average score of 4.42 for museums/museum-like Websites. We also obtained interesting results for a question regarding future work and the potential of NRT collaborative editing. While the NRT collaboration in general was received very positive, the participants seemed to appreciate workflows, in which they interfere with each other the least. Their most appreciated workflow was working separately on own parts of the story, closely followed by having different roles (such as *Story Scaffolder* and *Media Producer*). Organization seems to be a very important aspect of the collaborative editing, with one participant for example suggesting the use of a voice chat. The users were overall satisfied with the division of the *Viewer* website into widgets (one for the story and one for the 3D object).

Domain Expert Evaluation Results

The results of our second evaluation indicate that overall, the domain expert felt comfortable in using the editor after viewing the tutorial videos. He saw a use for the system in creating content for flexible online courses for student communities, rather than "explicit" storytelling. Although he appreciated the idea of a *Begin*, *Middle* and *End*, he wished for each of those components to entail more than just one *Media Object*. Especially in the case of a *Middle*, the expert suggested to use multiple consecutive middles, that allow for more extensive explanations. Another suggestion to create "excursions" inside the story, optional substories, that return to their entry point when finished. While this concept would be interesting to investigate, it is hardly compatible with MOD.

Overall, the expert appreciated the idea of also building stories around multiple objects and displaying parts of an object separately. He also liked the idea of having the possibility to hide certain parts of the story graph in order to reduce its complexity while editing. His suggestion was to make it possible to hide the subtree of any node.

7.2.4 Communication: Discussion and Lessons Learned

YouTell 2.0 was presented at a consortium meeting of the Virtus (56222-EPP-1-2015-1-EL-EPPKA3-PI-FORWARD) project, as showcase prototype for novel ways to achieve the scaffolding of vocational education training for CoPs and adult learners working in Tourism and Hospitality Services and Social Entrepreneurship. Its core dissemination represents the expert user evaluation. This envisions the usage of the tool once it grows over a beta stage for education purpose with students from the medicine domain.

YouTell 2.0 was easy to implement due to the metamodel-to-model editor generation. Its evaluation provides further evidence for the soundness and usability of our model-based

approach. This artifact also represents the realization of the reengineering of YouTell, providing NRT collaborative functionality to the asynchronous communication available in the older version.

There are also a number of open challenges identified after the realization of YouTell 2.0. For the current implementation, we limited the number of 3D objects which a story can be centered to just one. However, we see a high potential in allowing multiple objects and extending the approach to various multimedia objects. As Anatomy 2.0 also offers the possibility to only show certain areas of a 3D model (a certain body part, nerve, bone, etc.) the stories should be able to support such object areas as well. The orchestration using CAE of the available Anatomy 2.0, YouTell 2.0 and Vaptor is also of interest in the community context. We have not used the SyncMeta nudges or views during the realization of YouTell 2.0, but enabling these features and extending them for the storytelling prototype is very promising in terms of usage and usability improvements. This is due to the critique received on the strong enforcement of the MOD paradigm from the domain expert. In his opinion, there is a need for a more flexible approach that allows other types of storytelling usage (like online courses). In such a context, user nudges can play an important role in creating a coherent and compelling storytelling structure. Views can be also used to hide certain aspects of the graph, such as subtrees, representing the transitive closure of a node under the BME Split and Story Unit Connection relations. However, such a feature and semantic expressiveness of the metamodeling language needs to first be implemented into the SyncMeta framework.

7.3 Summary

This chapter presented the two evaluation studies performed to measure the usability of our overall approach and its main artifacts. Here, we gathered community developer's feedback and usage feedback from participants without technical knowledge, in simulated community settings using our test beds presented in the beginning of the dissertation. In order to showcase prototypes developed based on CAE principles or using the MDWE framework, we used two main case studies – developed community Web applications – that emerged based on elicited requirements from international or national projects in which we were involved and where we had access to professional CoPs. The developed applications are all widget-based, the design being consistent to the widgetizing methodology (identified functionality, clear-cut design, etc.). They are communicating via IWC and Yjs connectors at the frontend level and via RESTful interfaces with the backend microservices. A short overview of the status of the software artifacts and the relation to our proposed approach is given below:

Multimedia CIS Prior to the dissertation, the multimedia services available for the communities consist of a series of MPEG-7 backend annotation services and a cloud video transcoder Tethys cloud service (the most important services are described

in Chapter 2). They are composing the backend of the AnViAnno, SeViAnno and SeViAnno 2.0 video annotation and non-linear storytelling (YouTell) community applications. During the dissertation, the reengineering of the SeViAnno 2.0 widget-based Web application has been completed. SeViAnno 2.0 is available as a collection of lightweight annotation microservices and one RESTful cloud video transcoding microservice. The microservices are part of the Layers Box infrastructure [KKd+16]. Using the SeViAnno 2.0 annotation microservices are also used by Anatomy 2.0, the 3D object exploration environment. Moreover, adaptation and personalization of videos collected using SeViAnno 2.0 is achieved via the Vaptor system. The VAPTOR frontend and backend new components are generated using the CAE framework.

NRT Digital Storytelling YouTell 2.0 is an example of a widget-based Web application fully built using the SyncMeta framework. For creating YouTell 2.0, a MOD meta-model is modeled as a VLS and instantiated as the story editor. Together with additional widgets, interoperating with the editor, the resulting widgets and micro-services form the non-linear storytelling CIS. Apart from using SyncMeta's widgets, the YouTell 2.0 also reuses the annotation functionality of the Anatomy 2.0 prototype. Furthermore, the nudging feature of SyncMeta can be used in the future to support community members in the collaborative creation of stories.

Having a similar look and feel and providing uniform interfaces, the various components of SeViAnno 2.0, Anatomy 2.0 and YouTell 2.0 can be reused at the frontend and backend layers, in order to support heterogeneous CoPs. This is the case for example for the annotation microservices between SeViAnno 2.0 and Vaptor or Anatomy 2.0 and YouTell 2.0. Following the DevOpsUse approach, the applications can be extended, or components added or refactored. The extensibility can be socially triggered by community users using the Requirements Bazaar, Jira, or other social requirement elicitation platforms.

Since the microservices composing the various community applications use las2peer as platform, these are sending log data to a MobSOS microservice node, which in turn can be used to generate data analytics, for mining and data visualization purpose via the MobSOS framework. The components that have been generated using CAE can follow the application lifecycle described in Fig. 5.5. The integration of existing components that do not have a corresponding model (were not automatically generated using CAE) is not covered in this work, but it represents an interesting future work direction, for the automatic transformation of existing community services into CAE components. At the moment, a manual editing can be realized, but this requires a deep technical knowledge of the technologies used by CAE and development efforts.

These prototypes have been applied in the domains of Technology Enhanced Learning, Healthcare and Cultural Heritage. They were successfully designed, realized and disseminated. The evaluation studies performed with multiple users in collaborative settings, proved the applicability and soundness of our overall concepts contained within CAE, SyncMeta and Yjs and their suitability for CIS.

I never think of the future - it comes soon enough.

Albert Einstein (1879 - 1955)

Chapter 8

Conclusions and Future Work

In this dissertation, research has been conducted using the design science methodology, in the area of collaborative CIS design and development. Because of its interdisciplinary nature, the dissertation reports the results of investigations performed on several relevant topics from the computer science domain. Among these are conceptual modeling, computer-supported collaborative work, model-driven Web engineering, NRT shared editing, awareness and nudging, human computer interaction. In this section, we conclude by providing a summary of the obtained results, together with a list of main research contributions. We also offer some of the most important identified future work for our approach.

8.1 Summary of Results and Contributions

Overall, the requirements presented in Chapter 3 have been addressed and solutions have been presented for each of them. We described in the current work methodological and technical findings developed to support CoPs in engineering collaborative and social CIS (*CIS-R1*). To this end, we presented a methodology for reengineering single user applications into collaborative, community applications. A metamodel has been created based on empirical data and developer feedback. The metamodel has been refined and the resulting concepts have been generalized into an agile MDWE approach, based on the ATLAS methodology [Kla10b] (*CIS-R2*). CAE's cycle consists of a modeling and a coding phase. To support agility, both phases can be performed in NRT.

The NRT shared editing for CoPs is supported by a novel Web-based approach based on an algorithm belonging to the CRDT family. Its realization offers the needed flexibility to easily embed collaboration into existing Web applications and to design and develop new ones. We proved that the algorithm ensures consistency during synchronous collaborative edits and that its implementation, Yjs, has several advantages compared to other existing shared editing solutions; Available as a JavaScript library, it ensures consistency for various shared data types. It also supports multiple communication protocols, data persistence, late join,

garbage collection and offline editing. It offers a unique distributed, flexible and reliable approach towards NRT synchronization of various data types and formats, frequently used in Web development (JavaScript objects, DOM elements and XML, JSON, text, etc.). Its widespread adoption also evidences the need for such Web libraries.

Building on synchronous editing frameworks, we have presented a novel conceptual modeling approach, revisiting classical challenges from the modeling literature such as views and viewpoints, metamodel to model generation and visual modeling, from a new CSCW perspective. A pioneer in its domain, the resulting SyncMeta framework improves the collaboration aspects of conceptual modeling approaches and opens new research directions from a Web-based CIS perspective (*CIS-R3*). We also provide awareness and guidance features for the NRT actions of modelers (*CIS-R4*). The underlying platform, the ROLE SDK provides the technical support for community features, such as user management coupled with an OpenID Connect in-house provider, communities that act at a space level and synchronous communication based on the XMPP protocol (*CIS-R5*).

Thus, we offer a unique approach of NRT collaboration for view-based conceptual model editing on the Web using optimistic concurrency control mechanisms, combined with known techniques for views definition and code generation from the information systems domain. We have proved using various evaluation studies that our artifacts (cf. Fig. 8.2) are relevant and address needs and preferences of multiple stakeholders during the software creation processes. The SyncMeta and CAE frameworks were evaluated in group sessions in multiple settings. The evaluation results show that NRT collaboration between novices and experts working together during the design of CIS components is possible. Collaborative modeling and the NRT collaboration on complex data types can be successfully used in a MDWE approach to generate frontends and microservices. In our supported CIS infrastructure, we have chosen to work with P2P microservices and Web widgets, as they were at the time of our requirements elicitation and development phases the best suited technologies for the chosen communities and test beds. Furthermore, the microservice and widget-based architecture used in our approach allows for a high level of customization of the integrating parts of a CIS, computing environment, realization of collaboration, etc. They provide the flexibility needed by CoPs with respect to the CIS software architecture choices (*CIS-R7*). Apart from the collaborative aspects, CAE uses DevOpsUse technologies such as Docker and Jenkins for supporting rapid deployment, generating live previews of frontend components and continuous integration (*CIS-R6*).

Finally, all the various code components, frameworks and libraries have been developed as open source and are available on dedicated GitHub repositories (*CIS-R8*).

The main artifacts realized, together with the first iterations are presented in Fig. 8.1.

Based on the proposed approach, case studies and community prototypes were developed, that offer the technical support for learning, storytelling and other related practices for professional communities working in domains such as cultural heritage or technology enhanced learning. Our software artifacts were used by CoPs within the European projects

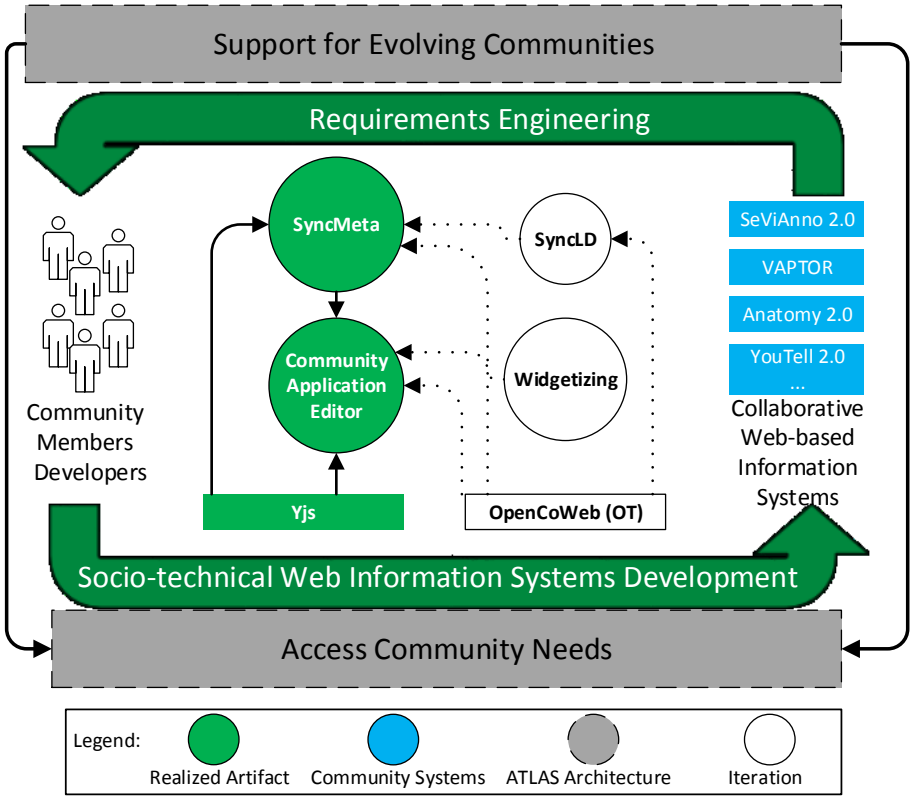


Figure 8.1: Artifacts (approach and software) achieved during the various iterations

presented in the introduction chapter of our dissertation. The main contributions and dissemination items of this work are summarized below:

Research Question 1 - Collaborative MDWE approach

- Widgetizing methodology ([NK15])
- Community Application Editor ([dND+16, dNKK16, dNKJ17])
- Collaborative 3D annotations for professional CoPs ([NTK15])
- Multimedia tagging on the Web ([NK14, KNK, NRK+14, KNSK17, KNK14, KRN+14])

Research Question 2 (RQ2) - CRDT-based approach for NRT shared editing as technical support for the agile MDWE approach realization

- NRT shared editing on multiple data types ([NJDK15, KNK15, NJDK16])

Research Question 3 (RQ3) - NRT collaborative conceptual modeling

- Domain-independent view-based visual modeling approach ([NDK13, DEN+14, DNE+15, NRD+16, NRD+18])

An overview of the various artifacts resulting or used in our work is presented in Fig. 8.2.

Conceptual Modeling The software artifacts include SyncLD and SyncMeta. SyncLD is developed as a domain dependent, widget-based Web tool, based on the OpenCoWeb OT library. The first three SyncMeta releases (each iteration of the software artifact resulted after an extension of the framework) are also based on the same OT library and have been developed based on the widgetizing methodology principles. Finally, the current SyncMeta artifact is fully based on Yjs and includes the views and nudging features.

Collaborative Editing Includes the OpenCoWeb open source third party library and the Yjs artifact contribution of the dissertation.

MDWE The various artifact iterations include the widgetizing methodology prototype, two versions of CAE based on the OT version of SyncMeta (SyncMeta II), the second including also partially functionality such as the live code editor, based on Yjs. The current CAE version is based on SyncMeta and uses Yjs for the live code editor and the generated collaborative elements.

Community-based Storytelling The artifacts are represented by the already existing YouTell non-linear storytelling application, which was reimplemented in YouTell 2.0 as a SyncMeta extension, using the Yjs library.

Multimedia Metadata The software artifacts include SeViAnno 2.0, a reengineering of the SeViAnno single-user Web application, Anatomy 2.0 and the Vaptor application.

8.2 Future Work

Each individual work identifies several threads for improvements and future work, in various aspects and dimensions of our research. Being of a more investigative nature, the usage of our proposed approach in professional communities opens up a wide range of customization and further research challenges.

In this dissertation, we cover only initial steps towards a complete integration for community users without technical knowledge into creating full-fledged Web applications. Our contribution can be enhanced by investigating further abstraction levels, human computer interaction practices or other methods that enable all CoP members to effectively specify requirements, design their own tools and implement and deploy complex Web applications. Research using CAE is currently continuing towards achieving a complete support for all

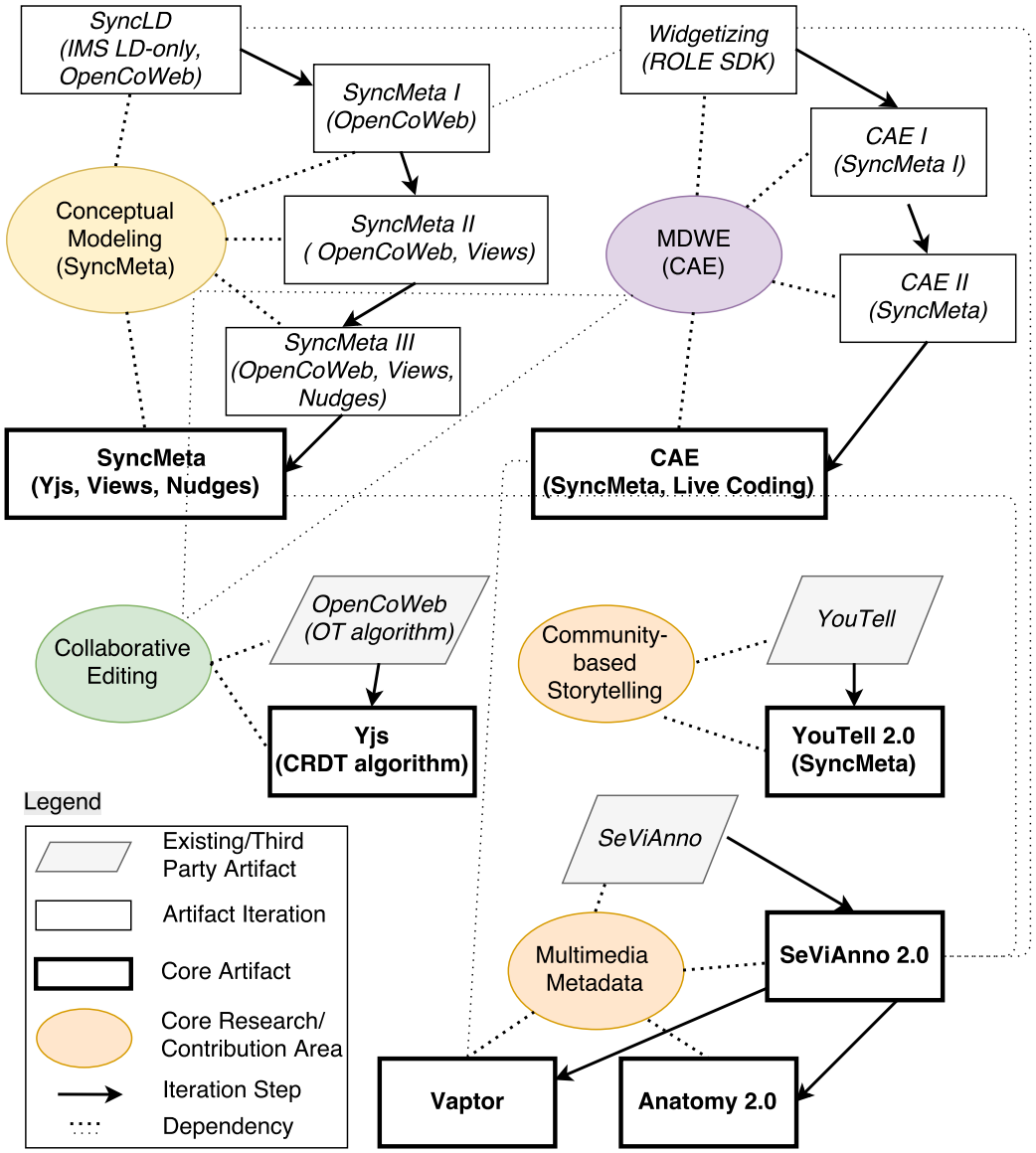


Figure 8.2: Artifacts used/developed and their interdependencies

community members and improving interactions between CoPs. Current works running at the Information Systems department cover end user friendly views during application modeling (wireframes, easily understandable by non-technical community members), to achieve a tighter integration of end users into the development practice. This can increase agility and the possibilities of CAE with respect to rapid prototyping techniques. With the wireframe, modeling and code views enabled, CAE benefits from various options to realize via the NRT synchronization a complete cyclic CIS engineering approach. Furthermore, other upcoming works will create an improved technological support for the P2P microser-

vice network that can be used to orchestrate the various available community services. The CAE approach can be used for further studies regarding the orchestration.

For achieving a powerful Web editor for community services, CAE's development support can be improved by involving external libraries management, compilation and run-time environments, etc. Such means can also enable the integration with further Web development tools established in practice.

CAE could also benefit from nudges and recommendations during the CIS modeling and generation process. The further study of the NRT collaboration patterns and the development cycle itself through the observation of the behaviour of various stakeholders, can offer valuable insights into the practice of targeted CoPs. Using this information, by extending our current nudging approach used during collaborative modeling, such support could recommend components to be used by community members, experts can be notified and guided to participate into current modeling or coding tasks, etc. These means can increase the social involvement and the sense of belonging within communities and act as a bridge between similar CoPs. Finally, research towards increasing the efficiency of CAE's collaborative processes and tasks should be performed. This can involve blending NRT collaboration and shared editing with asynchronous tasks such as versioning, adding new awareness capabilities during modeling and coding phases, etc.

Our technical realization for shared editing also raises several very interesting related work directions. A first one includes a more efficient garbage collection mechanism, in order to support operation removal while working offline and provide both functions simultaneously. Furthermore, the YATA algorithm and its implementation could be also optimized in order to further improve the scalability and reduce the time complexity. This can be achieved by tuning the approach towards using similar CRDT solutions (such as RGA) or through optimizations at the data type level in the library's implementation. Moreover, the Yjs framework can be enriched thorough the implementation of new data types and persistence mechanisms for increasing the usage scenarios and customization for the Web. Yjs can also be used to implement awareness in various shared editing scenarios, especially when combined with offline settings, synchronize state across Web widgets and frontend components [KRN+14], make Web pages more dynamic and enhance them with collaborative features, etc. Finally, embedding monitoring features could also help in the generation of collaboration traces and researching collaboration patterns from CoPs' members interactions.

Concerning the conceptual modeling approach, the semantics and expressiveness of the metamodels and the metamodel editor should be enhanced. Here, conditions on a view type to allow more complex queries and model perspectives can be added or support for model checking and further abstractions such as composition or aggregation represent only a starting point. In order to improve the feedback during collaborative modeling and the agility of SyncMeta, the automatic co-evolution of metamodels and models has to be studied. This is because currently we cannot synchronize the metamodel with the model editor and views in NRT. The instance generation step needs to be performed in advance.

Moreover, we plan to support alternative views, such as native HTML frontends for offering an improved end user experience during the agile Web information systems development. We are confident that these improvements will gear the framework towards widespread use in real-world information systems engineering projects. Finally, support for third party modeling tools can be achieved, including model import and export functionality, to reduce the platform dependence that SyncMeta conceptual models and encourage the usage of the framework in a larger scale, in both academia and industry.

In my personal opinion, Web-based NRT collaboration has not yet reached its true potential. In the context of digitalization, there are many useful scenarios that can be realized and from which professional CoPs, single users and organizations can take advantage from. With the help of stable and reliable protocols and the right architectural choices, CRDTs and synchronous shared editing can shape future database technologies and CSCW applications. Collaborative editors and other similar asynchronous and synchronous tools become usual features for Web applications. This trend has already started with the wide adoption of Google Docs and other tools for organizing work (calendars, todo lists, note taking tools, etc.). In a world where geographical boundaries are vanishing under the digital emerging ecosystems, the adoption of NRT collaboration increase also in information systems engineering, in online learning applications and in software development paradigms in general. I consider that these will become even more social and interconnected as nowadays. The collaboration technologies will make it easier to coordinate work across teams and reshape and reuse knowledge between Web communities. The ramifications of these trends into new business cases, human machine interaction, Web development, CIS and finally privacy and security are not fully known yet, but in the same time open the gate towards rewarding and interesting new research.

Bibliography

[AB91] Serge Abiteboul and Anthony Bonner. Objects and Views. In *ACM International Conference On Management Of Data (SIGMOD)*, pages 238–247. ACM New York, 1991.

[ABB+04] Roberto Acerbis, Aldo Bongio, Stefano Butti, Stefano Ceri, Fulvio Ciapessoni, Carlo Conserva, Piero Fraternali, and Giovanni Toffetti Carughi. WebRatio, an Innovative Technology for Web Application Development. In *Web Engineering*, volume 3140 of *Lecture Notes in Computer Science*, pages 613–614. Springer Berlin Heidelberg, 2004.

[ABBB15] Roberto Acerbis, Aldo Bongio, Marco Brambilla, and Stefano Butti. Model-Driven Development Based on OMG's IFML with WebRatio Web and Mobile Platform. In *Engineering the Web in the Big Data Era*, volume 9114 of *Lecture Notes in Computer Science*, pages 605–608. Springer International Publishing, 2015.

[Aca17] Basho Academy. Riak Documentation: *http://docs.basho.com/riak/kv/2.2.3/*, 2017.

[ACD+14] Carmelo Ardito, Maria Francesca Costabile, Giuseppe Desolda, Rosa Lanzilotti, Maristella Matera, Antonio Piccinno, and Matteo Picozzi. User-Driven Visual Composition of Service-Based Interactive Spaces. *Journal of Visual Languages & Computing*, 25(4):278–296, 2014.

[AK03] Colin Atkinson and Thomas Kühne. Model-Driven Development: A Metamodeling Foundation. *IEEE Software*, 20(5):36–41, 2003.

[akk17] akka.net. Akka.net Documentation: *http://getakka.net/index.html*, 2017.

[ALC08] László Angyal, László Lengyel, and Hassan Charaf. A Synchronizing Technique for Syntactic Model-Code Round-Trip Engineering. In *Proceedings of the 15th International Conference and Workshop on the Engineering of Computer-Based Systems*, ECBS '08, pages 463–472. IEEE Computer Society, 2008.

[AMOI13] Luc Andre, Stephane Martin, Gerald Oster, and Claudia-Lavinia Ignat. Supporting Adaptable Granularity of Changes for Massive-scale Collaborative Editing. In *Proceedings of the 9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom 2013)*, 2013.

[And08] Chris Anderson. *The Long Tail: Why the Future of Business is Selling Less of More*. Hyperion, New York, rev. and updated edition, 2008.

[ANIO+11] Mehdi Ahmed-Nacer, Claudia-Lavinia Ignat, Gérald Oster, Hyun-Gul Roh, and Pascal Urso. Evaluating CRDTs for Real-time Document Editing. In *Proceedings of the 11th ACM Symposium on Document Engineering*, pages 103–112. ACM New York, 2011.

[ANS12] ANSYS medini Technologies AG. medini QVT, 2012.

[APW03] Alexander Ardichvili, Vaughn Page, and Tim Wentling. Motivation and barriers to participation in virtual knowledge–sharing communities of practice. *Journal of Knowledge Management*, 7(1):64–77, 2003.

[Art11] W. Brian Arthur. *The Nature of Technology: What It Is and How It Evolves*. Free Press, New York, 1. ed. edition, 2011.

[ATS04] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A Survey of Peer-to-peer Content Distribution Technologies. *ACM Computing Surveys*, 36(4):335–371, 2004.

[Bal08] Rafael Ballagas. *Bringing Iterative Design to Ubiquitous Computing: Interaction Techniques Toolkits and Evaluation Methods*. PhD thesis, RWTH Aachen University, Göttingen, 2008.

[Bar01] Albert-László Barabási. The Physics of the Web. *Physics World*, 14:33–38, 2001.

[BBJN13] Adam Bergkvist, Daniel C. Burnett, Cullen Jennings, and Anant Narayanan. WebRTC 1.0: Real-time Communication Between Browsers: W3C Working Draft 10 September 2013, 2013.

[BCR04] Margaret Burnett, Curtis Cook, and Gregg Rothermel. End-User Software Engineering. *Communications of the ACM*, 47(9):53, 2004.

[BD91] John Seely Brown and Paul Duguid. Organizational Learning and Communities of Practice: Towards a Unified View of Working, Learning, and Innovation. *Organization Science*, 2(1):40–57, 1991.

[BD00] John Seely Brown and Paul Duguid. *The Social Life of Information*. Harvard Business Press, 2000.

[BDV10] Susan A. Brown, Alan R. Dennis, and Viswanath Venkatesh. Predicting Collaboration Technology Use: Integrating Technology Adoption and Collaboration Research. *Journal of Management Information Systems*, 27(2):9–54, 2010.

[Ben06] Yochai Benkler. *The Wealth of Networks: How Social Production Transforms Markets and Freedom*. Yale University Press, New Haven, CT and USA, 2006.

[BF14] Marco Brambilla and Piero Fraternali. *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2014.

[BH02] Karin Breu and Christopher Hemingway. Collaborative Processes and Knowledge Creation in Communities-of-Practice. *Creativity and Innovation Management*, 11(3):147–153, 2002.

[BK09] Marianne Busch and Nora Koch. MagicUWE – A CASE Tool Plugin for Modeling Web Applications. In *Proceedings of the 9th International Conference on Web Engineering*, volume 5648 of *ICWE'09*, pages 505–508. Springer, 2009.

[BMU14] Marco Brambilla, Andrea Mauri, and Eric Umuhoza. Extending the Interaction Flow Modeling Language (IFML) for Model Driven Development of Mobile Applications Front End. In *Mobile Web Information Systems*, volume 8640 of *Lecture Notes in Computer Science*, pages 176–191. Springer International Publishing, 2014.

[BPE+10] Jae Young Bang, Daniel Popescu, George Edwards, Nenad Medvidovic, Naveen Kulkarni, Girish M. Rama, and Srinivas Padmanabhuni. CoDesign: A Highly Extensible Collaborative Software Modeling Framework. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, pages 243–246, 2010.

[Bre00] Eric A. Brewer. Towards Robust Distributed Systems. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, page 7, 2000.

[Bru99] Bruno Latour. On Recalling ANT. In *Actor-Network Theory and After*, pages 15–25. Blackwell Publishers, 1999.

[BWZ15] Len Bass, Ingo Weber, and Liming Zhu. *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional, 2015.

[Cao12] Yiwei Cao. *Uncertainty Handling in Mobile Community Information Systems*. PhD thesis, RWTH Aachen University, Aachen, Germany, March 2012.

[CFB00] Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web Modeling Language (WebML): A Modeling Language for Designing Web sites. *Computer Networks*, 33(1):137–157, 2000.

[CGKP08] Yiwei Cao, Anna Glukhova, Ralf Klamma, and Zinayida Petrushyna. Getting to "Know" People on the Web 2.0. In *I-KNOW'08 and I-MEDIA'08, International Conferences on Knowledge Management and New Media Technology: Conference Proceedings*, J.UCS (Journal of Universal Computer Science) Proceedings, pages 169–176. Springer, 2008.

[CH06] Krzysztof Czarnecki and Simon Helsen. Feature-Based Survey of Model Transformation Approaches. *IBM Systems Journal*, 45(3):621–645, 2006.

[CHK+10] Yiwei Cao, Anna Hannemann, Ralf Klamma, Dejan Kovachev, and Dominik Renzel. Mobile Multimedia Management for Community-Aware Storytelling. In *2013 IEEE 14th International Conference on Mobile Data Management Proceedings*, pages 59–64. IEEE Computer Society, 2010.

[CKJ10] Yiwei Cao, Ralf Klamma, and Matthias Jarke. Mobile Multimedia Management for Virtual Campfire - The German Excellence Research Cluster UMIC. *International Journal on Computer Systems, Science and Engineering(IJCSSE)*, 25(3):251–265, 2010.

[CKJ11] Yiwei Cao, Ralf Klamma, and Matthias Jarke. The Hero's Journey - Template-Based Storytelling for Ubiquitous Multimedia Management. *Journal of Multimedia*, 6(2), 2011.

[CKK+15] Yiwei Cao, Dejan Kovachev, Ralf Klamma, Matthias Jarke, and Rynson W. H. Lau. Tagging Diversity in Personal Learning Environments. *Journal of Computers in Education*, 2(1):93–121, 2015.

[CKY+09] Yiwei Cao, Ralf Klamma, Yan Gao, Rynson W. H. Lau, and Matthias Jarke. A Web 2.0 Personal Learning Environment for Classical Chinese Poetry. In *SLKL09*, pages 98–107, 2009.

[CMPP08] Maria Francesca Costabile, Piero Mussio, Loredana Parasiliti Provenza, and Antonio Piccinno. End Users as Unwitting Software Developers. In *Proceedings of the 4th International Workshop on End-User Software Engineering*, WEUSE '08, pages 6–10. ACM, 2008.

[CRJ+10] Yiwei Cao, Dominik Renzel, Matthias Jarke, Ralf Klamma, Michael Lottko, Georgios Toubekis, and Michael Jansen. Well-Balanced Usability and Annotation Complexity in Interactive Video Semantization. In *4th International*

Conference on Multimedia and Ubiquitous Engineering, MUE 2010, pages 1–8, 2010.

[CSC+09] Marcelo Cataldo, Charles Shelton, Yongjoon Choi, Yun-Yin Huang, Vytesh Ramesh, Darpan Saini, and Liang-Yun Wang. CAMEL: A Tool for Collaborative Distributed Software Design. In *2009 Fourth IEEE International Conference on Global Software Engineering (ICGSE)*, pages 83–92, 2009.

[DA10] David Wang and Alex Mah Soren Lassen. Google Wave Operational Transformation: *http://www.waveprotocol.org/whitepapers/operational-transform*, 2010.

[Dal17] Richard Dallaway. WOOT with Scala.js: *https://github.com/d6y/wootjs*, 2017.

[DASH09] Prasun Dewan, Puneet Agarwal, Gautam Shroff, and Rajesh Hegde. Distributed side-by-side programming. In *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, pages 48–55, 2009.

[DB92] Paul Dourish and Bellotti Bellotti. Awareness and Coordination in Shared Workspaces. In *CSCW '92: Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work*, pages 107–114. ACM Press, 1992.

[DBJ05] Line Dubé, Anne Bourhis, and Réal Jacob. The Impact of Structuring Characteristics on the Launching of Virtual Communities of Practice. *Journal of Organizational Change Management*, 18(2):145–166, 2005.

[de 15] Peter de Lange. A Framework for Near Real-Time Modeling and Generation of Community Applications. Master's thesis, Chair for Information Systems and Databases, RWTH Aachen University, 2015.

[DEN+14] Michael Derntl, Stephan Erdtmann, Petru Nicolaescu, Ralf Klamma, and Matthias Jarke. Echtzeitmetamodellierung im Web-Browser. In *Modellierung 2014*, pages 65–80. Gesellschaft fur Informatik e.V., 2014.

[DFK02] Yvonne Dittrich, Christiane Floyd, and Ralf Klischewski. *Social Thinking–Software Practice*. MIT Press, Cambridge and MA, 2002.

[DGL+11] Stamatia Dasiopoulou, Eirini Giannakidou, Georgios Litos, Polyxeni Malasioti, and Yiannis Kompatsiaris. A Survey of Semantic Image and Video Annotation Tools. In *Knowledge-Driven Multimedia Information Extraction and Ontology Evolution*, volume 6050, pages 196–239. 2011.

[DKK+13] Michael Derntl, Ralf Klamma, István Koren, Miloš Kravčík, Petru Nicolaescu, Dominik Renzel, Kiarii Ngua, Jukka Purma, Graham Attwell, Owen Gray, Tobias Ley, Vladimir Tomberg, Christina Henry, Chris Whitehead, Dieter

Theiler, Christoph Trattner, Ronald K. Maier, Markus Manhart, Maria Schett, and Stefan Thalmann. Initial Architecture for Fast Small-Scale Deployment: D6.1, 2013.

[DKN+14] Michael Derntl, István Koren, Petru Nicolaescu, Dominik Renzel, and Ralf Klamma. Blueprint for Software Engineering in Technology Enhanced Learning Projects. In *Proceedings of the 9th European Conference on Open Learning and Teaching in Educational Communities*, volume 8719 of *LNCS*, pages 404–409. Springer Switzerland, 2014.

[DM03] William Hook DeLone and Ephraim R. McLean. The DeLone and McLean Model of Information Systems Success: A Ten-Year Update. *Journal of Management Information Systems*, 19(4):9–30, 2003.

[dND+16] Peter de Lange, Petru Nicolaescu, Michael Derntl, Matthias Jarke, and Ralf Klamma. Community Application Editor: Collaborative Near Real-Time Modeling and Composition of Microservice-based Web Applications. In *Modellierung 2016 Workshopband*, pages 123–127, 2016.

[DNE+15] Michael Derntl, Petru Nicolaescu, Stephan Erdtmann, Ralf Klamma, and Matthias Jarke. Near Real-Time Collaborative Conceptual Modeling on the Web. In *34th International Conference on Conceptual Modeling (ER 2015)*, volume 9381 of *Lecture Notes in Computer Science*, pages 344–357. Springer International Publishing, 2015.

[dNKJ17] Peter de Lange, Petru Nicolaescu, Ralf Klamma, and Matthias Jarke. Engineering Web Applications Using Real-Time Collaborative Modeling. In *Proceedings of the 23rd International Conference on Collaboration and Technology*, volume 10391 of *Lecture Notes in Computer Science*, pages 213–228, 2017.

[dNKK16] Peter de Lange, Petru Nicolaescu, Ralf Klamma, and István Koren. DevOpsUse for Rapid Training of Agile Practices Within Undergraduate and Startup Communities. In *Proceedings of the 11th European Conference on Technology Enhanced Learning (EC-TEL 2016)*, volume 9891 of *Lecture Notes in Computer Science*, pages 570–574. Springer International Publishing, 2016.

[DNO11] Michael Derntl, Susanne Neumann, and Petra Oberhuemer. Propelling Standards-based Sharing and Reuse in Instructional Modeling Communities: The Open Graphical Learning Modeler (OpenGLM). In *Proceedings of 11th IEEE International Conference on Advanced Learning Technologies (ICALT)*, pages 431–435. IEEE, 2011.

[DNTK13] Michael Derntl, Petru Nicolaescu, Bezunesh Alemu Terkik, and Ralf Klamma. SynC-LD: Synchronous Collaborative IMS Learning Design Authoring on

the Web. In *Proceedings of the 8th European Conference on Technology Enhanced Learning*, volume 8095 of *LNCS*, pages 540–543. Springer-Verlag, 2013.

[DR93] Prasun Dewan and John Riedl. Toward Computer-Supported Concurrent Software Engineering. *Computer*, 26(1):17–27, 1993.

[DRN+15] Michael Derntl, Dominik Renzel, Petru Nicolaescu, István Koren, and Ralf Klamma. Distributed Software Engineering in Collaborative Research Projects. In *Proceedings 2015 IEEE 10th International Conference on Global Software Engineering (ICGSE 2015)*, pages 105–109. IEEE Computer Society, 2015.

[DSL02] Aguido Horatio Davis, Chengzheng Sun, and Junwei Lu. Generalizing Operational Transformation to the Standard General Markup Language. In *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work*, CSCW '02, pages 58–67. ACM, 2002.

[Eck06] Penelope Eckert. Communities of Practice. *Encyclopedia of Language and Linguistics*, 2(2006):683–685, 2006.

[EDZC09] Liliane Esnault, Amaury Daele, Romain Zeiliger, and Bernadette Charlier. Creating an Innovative Palette of Services for Communities of Practice with Participatory Design. In *Proceedings of the 4th European Conference on Technology Enhanced Learning*, volume 5794 of *Lecture Notes in Computer Science*, pages 304–309. Springer Berlin Heidelberg, 2009.

[EEHT05] Karsten Ehrig, Claudia Ermel, Stefan Hänsgen, and Gabriele Taentzer. Generation of Visual Editors as Eclipse Plug-ins. In *Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering*, pages 134–143, 2005.

[EG89] Clarence A. Ellis and Simon J. Gibbs. Concurrency Control in Groupware Systems. *SIGMOD Rec*, 18(2):399–407, 1989.

[EJ01] Robert Esser and Jörn W. Janneck. A Framework for Defining Domain-Specific Visual Languages. In *Workshop on Domain Specific Visual Languages, ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA-2001)*, 2001.

[Erl05] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, 2005.

[FAGS11] Zameer Fatima, Ankit Agarwal, Gaurav Gupta, and Mayank Sharma. Group Editer Using Graphical Operational Transformation. In *INDIACom - 2011; 5th National Conference on Computing For Nation Development*, 2011.

[Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-Based Software Architectures*. PhD thesis, University of California, Irvine, CA, USA, 2000.

[Fis01] Gerhard Fischer. User Modeling in Human-Computer Interaction. In *The Journal of Personalization Research*, volume 11, pages 65–86. 2001.

[FK13] Hans-Georg Fill and Dimitris Karagiannis. On the Conceptualisation of Modelling Methods Using the ADOxx Meta Modelling Platform. *Enterprise Modelling and Information Systems Architectures*, 8(1), 2013.

[Fow09] Martin Fowler. *UML distilled: A brief guide to the standard object modeling language*. The Addison-Wesley Object Technology Series. Addison-Wesley, Boston, Mass., 3. ed., 15. print edition, 2009.

[Fox00] Stephen Fox. Communities Of Practice, Foucault And Actor-Network Therory. *Journal of Management Studies*, 37(6):853–868, 2000.

[FP10] Martin Fowler and Rebecca Parsons. *Domain Specific Languages*. Addison-Wesley Professional Computing Series. Addison-Wesley, Upper Saddle River, NJ, 1st edition, 2010.

[FPK+12] Klaus Fischer, Dima Panfilenko, Julian Krumeich, Marc Born, and Philippe Desfray. Viewpoint-Based Modeling - Towards Defining the Viewpoint Concept and Implications for Supporting Modeling Tools. In *International Workshop on Enterprise Modelling*, Lecture Notes in Informatics (LNI), pages 123–136. Springer, 2012.

[Fra09] Neil Fraser. google-mobwrite: Real-time Synchronization and Collaboration Service - Google Project Hosting. [Online] http://code.google.com/p/google-mobwrite/, 2009.

[GBB12] Thomas Goldschmidt, Steffen Becker, and Erik Burger. Towards a Tool-Oriented Taxonomy of View-Based Modelling. In *Modellierung 2012*. 2012.

[GBR12] Jesús Gallardo, Crescencio Bravo, and Miguel A. Redondo. A Model-Driven Development Method for Collaborative Modeling Tools. *Journal of Network and Computer Applications*, 35(3):1086–1105, 2012.

[GCHH+12] Orlena Gotel, Jane Cleland-Huang, Jane Huffman Hayes, Andrea Zisman, Alexander Egyed, Paul Grünbacher, Alex Dekhtyar, Giuliano Antoniol, and Jonathan Maletic. The Grand Challenge of Traceability (v1.0). In *Software and Systems Traceability*, pages 343–409. Springer, 2012.

[Ger06] Gerlicher, Ansgar. A Framework for Real-time Collaborative Engineering in the Automotive Industries. In *Proceedings of the Third International Conference on Cooperative Design, Visualization, and Engineering*, CDVE'06, pages 164–173. Springer-Verlag, 2006.

[GH09] Holger Giese and Stephan Hildebrandt. *Efficient Model Synchronization of Large-Scale Models*, volume 28 of *Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam*. Universitätsverlag Potsdam, Potsdam, 2009.

[GLM] Max Goldman, Greg Little, and Robert C. Miller. Real-Time Collaborative Coding in a Web IDE. In *Proceedings of the 24th ACM Symposium on User Interface Software and Technology*, pages 155–164.

[GM94] Saul Greenberg and David Marwood. Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, pages 207–217, 1994.

[Goe06] Stefan Goessner. Converting between XML and JSON, 2006.

[GP16] Andrea Gallidabino and Cesare Pautasso. The Liquid.js Framework for Migrating and Cloning Stateful Web Components across Multiple Devices. In *WWW'16 Companion*, pages 183–186, 2016.

[Gri17] Victor Grishchenko. Swarm.js: *http://swarmjs.github.io/*, 2017.

[Gro13] Tom Gross. Supporting Effortless Coordination: 25 Years of Awareness Research. *Computer Supported Cooperative Work*, 22(4-6):425–474, 2013.

[Gru94] Jonathan Grudin. Computer-Supported Cooperative Work: History and Focus. *Computer*, 27(5):19–26, 1994.

[GVD+11] Sten Govaerts, Katrien Verbert, Daniel Dahrendorf, Carsten Ullrich, Manuel Schmidt, Michael Werkle, Arunangsu Chatterjee, Alexander Nussbaumer, Dominik Renzel, Maren Scheffel, Martin Friedrich, Jose Luis Santos, Erik Duval, and Effie Law. Towards Responsive Open Learning Environments: The ROLE Interoperability Framework. In *Proceedings of the 6th European Conference of Technology Enhanced Learning (EC-TEL 2011)*, volume 6964 of *LNCS*, pages 125–138. Springer Berlin Heidelberg, 2011.

[GW06] Holger Giese and Robert Wagner. Incremental Model Synchronization with Triple Graph Grammars. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems*, volume 4199, pages 543–557. Springer, 2006.

[GWG97] Hans-Werner Gellersen, Robert Wicke, and Martin Gaedke. WebComposition: An Object-Oriented Support System for the Web Engineering Lifecycle. *Computer Networks and ISDN Systems*, 29(8-13):1429–1437, 1997.

[HA13] Sonja Hollins-Alexander. *Online Professional Development Through Virtual Learning Communities: The Learner-Learner Model*. Insight Education Group, Inc., 2013.

[Han14] Anna Hannemann. *Requirements Management in Community-Oriented Software Development*. PhD thesis, RWTH Aachen University, Aachen, 2014.

[HC10] Alan Hevner and Samir Chatterjee. *Design Research in Information Systems*, volume 22 of *Integrated Series in Information Systems*. Springer Science+Business Media LLC, Boston, MA, 2010.

[HD08] Rajesh Hegde and Prasun Dewan. Connecting Programming Environments to Support Ad-Hoc Collaboration. In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pages 178–187, 2008.

[Hei14] Matthias Heinrich. *Enriching Web Applications Efficiently with Real-Time Collaboration Capabilities*, volume Vol. 1 of *Doctoral Dissertations in Web Engineering and Web Science*. Univ.-Verl, Chemnitz, 2014.

[HGSG12] Matthias Heinrich, Franz Josef Grüneberger, Thomas Springer, and Martin Gaedke. Enriching Web Applications with Collaboration Support Using Dependency Injection. In *12th International Conference on Web Engineering (ICWE 2012)*, volume 7387 of *Lecture Notes in Computer Science*, pages 473–476. Springer, 2012.

[HHWM92] William C. Hill, James D. Hollan, Dave Wroblewski, and Tim McCandless. Edit Wear and Read Wear. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3–9, 1992.

[Hic11] Ian Hickson. HTML5. [Online] https://www.w3.org/TR/2011/WD-html5-20110525/, 2011.

[HK92] Austin Henderson and Morten Kyng. There's No Place Like Home: Continuing Design in Use. In *Design at Work*, pages 219–240. L. Erlbaum Associates Inc., 1992.

[HL10] Lile Hattori and Michele Lanza. Syde: A Tool for Collaborative Software Development. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, volume 2, page 235, 2010.

[HLR08] Thomas Hettel, Michael Lawley, and Kerry Raymond. Model Synchronisation: Definitions for Round-Trip Engineering. In *Proceedings of the First International Conference on Model Transformations*, pages 31–45. Springer, 2008.

[HLVA+06] Davinia Hernández-Leo, Villasclaras-Fernández, Eloy D., Asensio-Pérez, Juan I., Dimitriadis, Yannis, Jorrín-Abellán, Iván M., Ruiz-Requies, Inés, and Bartolomé, Rubia-Avi. COLLAGE: A Collaborative Learning Design Editor Based on Patterns. *Educational Technology and Society*, 9(1):58–71, 2006.

[HMPR04] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004.

[HMTK04] Hans Hummel, Jocelyn Manderveld, Colin Tattersall, and Rob Koper. Educational Modelling Language and Learning Design: New Opportunities for Instructional Reusability and Personalised Learning. *International Journal of Learning Technology*, 1(1):111–126, 2004.

[HPv05] Hoppenbrouwers, Stijn J. B. A, Proper, Henderik A. (Erik), and van der Weide, Theo P. A Fundamental View on the Process of Conceptual Modeling. In *Proceedings of the 24th International Conference on Conceptual Modeling (ER 2005)*, pages 128–143. Springer, 2005.

[HSH+02] Christian Heath, Marcus Sanchez Svensson, Jon Hindmarsh, Paul Luff, and Dirk Vom Lehn. Configuring Awareness. *Computer Supported Cooperative Work*, 11(3):317–347, 2002.

[Hüt12] Michael Hüttermann. *DevOps for Developers: The Expert's Voice in Web Development*. Apress, Berkeley, CA, USA, 2012.

[IL84] Blake Ives and Gerard P. Learmonth. The information system as a competitive weapon. *Communications of the ACM*, 27(12):1193–1201, 1984.

[IMS03] IMS Global Learning Consortium. Learning Design Specification. [Online] http://www.imsglobal.org/learningdesign/, 2003.

[IN02] Claudia Ignat and Moira C. Norrie. Tree-based Model Algorithm for Maintaining Consistency in Real-Time Collaborative Editing Systems. In *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work (CSCW '02)*. ACM, 2002.

[IN03] Claudia-Lavinia Ignat and Moira C. Norrie. Customizable Collaborative Editor Relying on treeOPT Algorithm. In *Proceedings of the Eighth Conference on European Conference on Computer Supported Cooperative Work*, ECSCW'03, pages 315–334. Kluwer Academic Publishers, 2003.

[IN08] Claudia-Lavinia Ignat and Moira C. Norrie. Multi-level Editing of Hierarchical Documents. *Computer Supported Cooperative Work*, 17(5-6):423–468, 2008.

[IPN+13] Erik Isaksson, Matthias Palmér, Ambjörn Naeve, Daniel Dahrendorf, Nils Faltin, Christina Höbelt, Volker Zimmermann, Martin Friedrich, Maren Scheffel, Dominik Renzel, Evgeny Bogdanov, Li Na, Katrien Verbert, Sten Govaerts, Felix Mödritscher, Alexander Mikroyannidis, Carsten Ullrich, and Alexander Nussbaumer. Personal Learning Service Bundles (Update M48), 2013.

[ISO] ISO/IEC/IEEE. Systems and software engineering - Architecture Descriptions in ISO/IEC/IEEE 42010.

[Jäg02] Ludwig Jäger. Transkriptivität - Zur medialen Logik der kulturellen Semantik. In *Transkribieren - Medien/Lektüre*, pages 19–41. Wilhelm Fink Verlag, 2002.

[JGJ+95] Matthias Jarke, R. Gallersdörfer, Manfred A. Jeusfeld, Martin Staudt, and Stefan Eherer. ConceptBase - a Deductive Object Base Manager for Meta Data Management. *Journal of Intelligent Information Systems*, 4(2):167–192, 1995.

[JJKS08] Ludwig Jäger, Matthias Jarke, Ralf Klamma, and Marc Spaniol. Transkriptivität: Operative Medientheorien als Grundlage von Informationssystemen für die Kulturwissenschaften. *Informatik Spektrum*, 31(1):21–29, 2008.

[JK02] Matthias Jarke and Ralf Klamma. Metadata and Cooperative Knowledge Management. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering, May 27-31, 2002, Toronto, Ontario, Canada*, volume 2348 of *Lecture Notes in Computer Science*, pages 4–20. Springer-Verlag, 2002.

[JK05] Matthias Jarke and Ralf Klamma. Transkriptivität als informatisches Designprinzip: Mediale Spuren in rechnergestützten Entwicklungsprozessen. In *Spuren Lektüren*, pages 105–120. Wilhelm Fink Verlag, 2005.

[JLB12] Jordan Janeiro, Stephan Lukosch, and Frances Brazier. Elastic collaboration support: From workflow-based to emergent collaboration. In *Proceedings of the 17th ACM International Conference on Supporting Group Work*, GROUP '12, pages 317–320, New York, NY, USA, 2012. ACM.

[JMSV92] Matthias Jarke, J. Mylopoulos, J. W. Schmidt, and Y. Vassiliou. DAIDA: an environment for evolving information systems. *ACM Transactions on Information Systems*, 10(1):1–50, 1992.

[JR88] Matthias Jarke and Thomas Rose. Managing knowledge about information system evolution. In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, SIGMOD '88, pages 303–311. ACM, 1988.

[Kar15] Dimitris Karagiannis. Agile Modeling Method Engineering. In *the 19th Panhellenic Conference*, pages 5–10, 2015.

[KB98] Finn Kensing and Jeanette Blomberg. Participatory Design: Issues and Concerns. *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*, 7(3-4):167–185, 1998.

[KGK13] István Koren, Andreas Guth, and Ralf Klamma. Shared Editing on the Web: A Classification of Developer Support Libraries. In *Proceedings of the 9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom 2013)*, pages 468–477, 2013.

[KJ08] Ralf Klamma and Matthias Jarke. Mobile Social Software for Professional Communities. *UPGRADE*, IX(3):37–43, 2008.

[KK02] Nora Koch and Andreas Kraus. The Expressive Power of UML-based Web Engineering. In *Second International Workshop on Web-oriented Software Technology (IWWOST02)*, volume 16, pages 105–119, 2002.

[KK03] Nora Koch and Andreas Kraus. Towards a common Metamodel for the Development of Web Applications. In *3rd International Conference on Web Engineering*, pages 497–506. Springer, 2003.

[KK11] Miloš Kravčík and Ralf Klamma. On Psychological Aspects of Learning Environments Design. In *Towards Ubiquitous Learning*, volume 6964 of *LNCS*, pages 436–441. Springer Berlin Heidelberg, 2011.

[KK12] Nora Koch and Sergej Kozuruba. Requirements Models as First Class Entities in Model-Driven Web Engineering. In *Current Trends in Web Engineering*, volume 7703 of *Lecture Notes in Computer Science*, pages 158–169. Springer Berlin Heidelberg, 2012.

[KK14] Dejan Kovachev and Ralf Klamma. Supporting Practices in Professional Communities Using Mobile Cloud Services. In *Cloud Computing and Digital Media: Fundamentals, Techniques, and Applications*, pages 47–81. Chapman and Hall/CRC, 2014.

[KKd+16] Ralf Klamma, István Koren, Peter de Lange, Petru Nicolaescu, Dominik Renzel, and Mohsen Shahriari. Learning Layers Results: Infrastructure. [Online] http://results.learning-layers.eu/infrastructure/, 2016.

[KKK07] Andreas Kraus, Alexander Knapp, and Nora Koch. Model-Driven Generation of Web Applications in UWE. In *3rd International Workshop on Model-Driven Web Engineering (MDWE)*, volume 261. CEUR-WS, 2007.

[KKK09] Christian Kroiss, Nora Koch, and Alexander Knapp. UWE4JSF: A Model-Driven Generation Approach for Web Applications. In *Proceedings of the 9th International Conference on Web Engineering*, ICWE'09, pages 493–496. Springer, 2009.

[KKN+15] Ralf Klamma, István Koren, Petru Nicolaescu, Dominik Renzel, Miloš Kravčík, Mohsen Shahriari, Michael Derntl, Gilbert Peffer, and Raymond Elferink. DevOpsUse - Scaling Continuous Innovation: Project Deliverable, 2015.

[KKZB08] Nora Koch, Alexander Knapp, Gefei Zhang, and Hubert Baumeister. UML-based Web Engineering: An Approach based on Standards. In *Web Engineering: Modelling and Implementing Web Applications*, pages 157–191. Springer, 2008.

[KKZH04] Alexander Knapp, Nora Koch, Gefei Zhang, and Hanns-Martin Hassler. Modeling Business Processes in Web Applications with ArgoUWE. In *Proceedings of the 7th International Conference on the Unified Modeling Language*, pages 69–83. 2004.

[Kla10a] Ralf Klamma. *Social Software and Community Information Systems*. PhD thesis, RWTH Aachen University, Aachen, Germany, 2010.

[Kla10b] Ralf Klamma. Werkzeuge und Modelle für die Übergreifende Untersuchung von Social Software. *i-com*, 9(3):12–20, 2010.

[Kla13] Ralf Klamma. Community Learning Analytics – Challenges and Opportunities. In *Advances in Web-Based Learning: ICWL 2013*, volume 8167 of *LNCS*, pages 284–293. Springer, 2013.

[Kla15] Ralf Klamma. Near-Real-Time Social Computing. *IEEE Computer*, 48(9):90–92, 2015.

[KLRT13] Steven Kelly, Kalle Lyytinen, Matti Rossi, and Juha-Pekka Tolvanen. MetaEdit+ at the Age of 20. In *Seminal Contributions to Information Systems Engineering*, pages 131–137. Springer, 2013.

[KMR+11] Andrew J. Ko, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, Susan Wiedenbeck, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, and Henry Lieberman. The state of the art in end-user software engineering. *ACM Computing Surveys*, 43(3):1–44, 2011.

[KNK] Miloš Kravčík, Petru Nicolaescu, and Ralf Klamma. Informal Learning at the Workplace via Adaptive Video. In *UMAP, ProS: Workshop on UMAP Projects Synergy*, volume 1181, pages 35–38.

[KNK14] Dejan Kovachev, Petru Nicolaescu, and Ralf Klamma. Mobile Real-Time Collaboration for Semantic Multimedia. *Mobile Networks and Applications*, 19(5):635–648, 2014.

[KNK15] István Koren, Petru Nicolaescu, and Ralf Klamma. Collaborative Drawing Annotations on Web Videos. In *Engineering the Web in the Big Data Era*, volume 9114 of *LNCS*, pages 671–674. Springer, 2015.

[KNSK17] Miloš Kravčík, Petru Nicolaescu, Aarij Siddiqui, and Ralf Klamma. Adaptive Video Techniques for Informal Learning Support in Workplace Environments. In *First International Symposium in Emerging Technologies for Education*, volume 10108 of *Lecture Notes in Computer Science*, pages 533–543. 2017.

[KO04] Rob Koper and Bill Olivier. Representing the Learning Design of Units of Learning. *Educational Technology and Society*, 7(3):97–111, 2004.

[KOB07] John Krogstie, Andreas Lothe Opdahl, and Sjaak Brinkkemper, editors. *Conceptual Modelling in Information Systems Engineering*. Springer, Berlin Heidelberg, 2007.

[Kov14] Dejan Kovachev. *Mobile multimedia services in the cloud*. PhD thesis, RWTH Aachen University, Aachen, 2014.

[Koz06] Melita Kozina. Evaluation of ARIS and Zachman Frameworks as Enterprise Architectures. *Journal of Information and Organizational Sciences*, 30(1), 2006.

[KPZM09] Nora Koch, Matthias Pigerl, Gefei Zhang, and Tatiana Morozova. Patterns for the Model-Based Development of RIAs. In *Proceedings of the 9th International Conference on Web Engineering (ICWE'09)*, pages 283–291. 2009.

[KRdJ16] Ralf Klamma, Dominik Renzel, Peter de Lange, and Holger Janßen. las2peer - A Primer. ResearchGate, 2016.

[KRKC10] Dejan Kovachev, Dominik Renzel, Ralf Klamma, and Yiwei Cao. Mobile Community Cloud Computing: Emerges and Evolves. In *Proceedings of the First International Workshop on Mobile Cloud Computing (MCC)*, pages 393–395. IEEE, 2010.

[KRN+14] Dejan Kovachev, Dominik Renzel, Petru Nicolaescu, István Koren, and Ralf Klamma. DireWolf: A Framework for Widget-based Distributed User Interfaces. *Journal of Web Engineering*, 13(3&4):203–222, 2014.

[KRNK13] Dejan Kovachev, Dominik Renzel, Petru Nicolaescu, and Ralf Klamma. DireWolf - Distributing and Migrating User Interfaces for Widget-Based Web Applications. In *13th International Conference on Web Engineering (ICWE 2013)*, volume 7977 of *LNCS*, pages 99–113. Springer, 2013.

[KSCJ06] Ralf Klamma, Marc Spaniol, Yiwei Cao, and Matthias Jarke. Pattern-Based Cross Media Social Network Analysis for Technology Enhanced Learning in Europe. In *First European Conference on Technology Enhanced Learning: Innovative Approaches to Learning and Knowledge Sharing*, volume 4227 of *LNCS*, pages 242–256. Springer, 2006.

[KW07a] Maged N. Kamel Boulos and Steve Wheeler. The Emerging Web 2.0 Social Software: An Enabling Suite of Sociable Technologies in Health and Health Care Education. *Health information and libraries journal*, 24(1):2–23, 2007.

[KW07b] Stephan Kurpjuweit and Robert Winter. Viewpoint-based Meta Model Engineering. In *Proceedings of the 2nd International Workshop on Enterprise Modelling and Information Systems Architectures*, volume 119 of *GI-Edition / Proceedings*, pages 145–158. Ges. für Informatik, 2007.

[Lan13] Mark Lankhorst. *Enterprise Architecture at Work: Modelling, Communication, and Analysis*. Springer, 3rd edition, 2013.

[LCK+16] Yanyan Li, Maiga Chang, Milos Kravcik, Elvira Popescu, Ronghuai Huang, and Kinshuk Nian-Shing Chen, editors. *State-of-the-Art and Future Directions of Smart Learning*. Lecture notes in educational technology. Springer, 2016.

[LDS04] Rui Li, Du Li, and Chengzheng Sun. A Time Interval Based Consistency Control Algorithm for Interactive Groupware Applications. In *Proceedings of the Parallel and Distributed Systems, Tenth International Conference*, ICPADS '04, pages 429–436. IEEE Computer Society, 2004.

[LKB11] Stephan Lukosch, Michael Klebl, and Tanja Buttler. Utilizing verbally told stories for informal knowledge management. *Group Decision and Negotiation*, 20(5):615–642, 2011.

[LPW06] Henry Lieberman, Fabio Paternò, and Volker Wulf. End-User Development: An Emerging Paradigm. In *End User Development*, volume 9, pages 1–8. 2006.

[LS01] Eric L. Lesser and John Storck. Communities of Practice and Organizational Performance. *IBM Systems Journal*, 40(4):831–841, 2001.

[LV02] Juan de Lara and Hans Vangheluwe. AToM3: A Tool for Multi-formalism and Meta-modelling. In *Fundamental Approaches to Software Engineering*,

volume 2306 of *Lecture Notes in Computer Science*, pages 174–188. Springer Berlin Heidelberg, 2002.

[LW91] Jean Lave and Etienne Wenger. *Situated Learning: Legimate Peripheral Participation*. Cambridge University Press, Cambridge, UK, 1991.

[LXZS14] Yang Liu, Yi Xu, Shao Jie Zhang, and Chengzheng Sun. Formal Verification of Operational Transformation. In *FM 2014: Formal Methods*, volume 8442 of *Lecture Notes in Computer Science*, pages 432–448. Springer International Publishing, 2014.

[Mac90] Wendy E. Mackay. Patterns of sharing customizable software. In *Proceedings of the 1990 ACM Conference on Computer-supported Cooperative Work*, pages 209–221, 1990.

[Mar82] James Martin. *Application development without programmers*. Prentice-Hall, Englewood Cliffs, N.J., 3. [print.] edition, 1982.

[MBd07] Alexandre Pereira Meire, Marcos Borges, and de Araújo, Renata Mendes. Supporting multiple viewpoints in collaborative graphical editing. *Multimedia Tools and Applications*, 32(2):185–208, 2007.

[MBJK90] J. Mylopoulos, A. Borgida, Matthias Jarke, and M. Koubarakis. Telos - a Language for Representing Knowledge about Information Systems. *ACM Transactions on Information Systems*, 8(4):352 – 362, 1990.

[MF00] Molly McLure Wasko and Samer Faraj. "It Is What One Does": Why People Participate and Help Others in Electronic Communities of Practice. *The Journal of Strategic Information Systems*, 9(2-3):155–173, 2000.

[MFB+08] Brice Morin, Franck Fleurey, Nelly Bencomo, Jean-Marc Jézéquel, Arnor Solberg, Vegard Dehlen, and Gordon Blair. An Aspect-Oriented and Model-Driven Approach for Managing Dynamic Variability. In *Proceedings of the First International Conference on Model Transformations*, pages 782–796. Springer, 2008.

[Min07] Mark Minas. Generating Meta-Model-Based Freehand Editors. *Electronic Communications of the EASST*, 1, 2007.

[MKB06] Brad A. Myers, Andrew J. Ko, and Margaret M. Burnett. Invited research overview. In *CHI '06 extended abstracts*, page 75, 2006.

[MRV08] Nathalie Moreno, José Raúl Romero, and Antonio Vallecillo. An Overview Of Model-Driven Web Engineering and the Mda. In *Web Engineering: Modelling and Implementing Web Applications*, Human-Computer Interaction Series, pages 353–382. Springer-Verlag London Limited, 2008.

[MS03] I. Scott MacKenzie and R. William Soukoreff. Phrase Sets for Evaluating Text Entry Techniques. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems*, pages 754–755, 2003.

[MSAM10] Peter Millard, Peter Saint-Andre, and Ralph Meijer. XEP-0060: Publish-subscribe. *XMPP Standards Foundation*, 1:13, 2010.

[MSUW02] Stephen J. Mellor, Kendall Scott, Axel Uhl, and Dirk Weise. Model-Driven Architecture. In *Proceedings of the 8th International Conference on Object-Oriented Information Systems*, pages 290–297. Springer, 2002.

[Mv06] Tom Mens and Pieter van Gorp. A Taxonomy of Model Transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142, 2006.

[MWW92] Michael J. Muller, Daniel M. Wildman, and Ellen A. White. Taxonomy of Participatory Design Practices. In *Posters and short talks of the 1992 SIGCHI Conference in Human Factors in Computing Systems*, page 34. ACM New York, 1992.

[NCDL95] David A. Nichols, Pavel Curtis, Michael Dixon, and John Lamping. High-latency, Low-bandwidth Windowing in the Jupiter Collaboration System. In *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*, UIST '95, pages 111–120. ACM, 1995.

[NDK13] Petru Nicolaescu, Michael Derntl, and Ralf Klamma. Browser-based Collaborative Modeling in Near Real-Time. In *Proceedings of the 9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom 2013)*, pages 335–344, 2013.

[New15] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly, Sebastopol, CA, USA, first edition edition, 2015.

[NJ99] Hans W. Nissen and Matthias Jarke. Repository support for multi-perspective requirements engineering. *Information Systems*, 24(2):131–158, 1999.

[NJDK15] Petru Nicolaescu, Kevin Jahns, Michael Derntl, and Ralf Klamma. Yjs: A Framework for Near Real-Time P2P Shared Editing on Arbitrary Data Types. In *Proceedings of the 15th International Conference on Web Engineering (ICWE 2015)*, volume 9114 of *LNCS*, pages 675–678. Springer, 2015.

[NJDK16] Petru Nicolaescu, Kevin Jahns, Michael Derntl, and Ralf Klamma. Near Real-Time Peer-to-Peer Shared Editing on Extensible Data Types. In *GROUP 2016*. ACM, 2016.

[NJJ+96] Hans W. Nissen, Manfred A. Jeusfeld, Matthias Jarke, Georg Zemanek, and Harald Huber. Managing Multiple Requirements Perspectives with Meta Models. *IEEE Software*, pages 37–48, 1996.

[NK14] Petru Nicolaescu and Ralf Klamma. SeViAnno 2.0: Web-Enabled Collaborative Semantic Video Annotation Beyond the Obvious. In *12th International Workshop on Content-Based Multimedia Indexing 2014*, pages 1–6, 2014.

[NK15] Petru Nicolaescu and Ralf Klamma. A Methodology and Tool Support for Widget-based Web Application Development. In *Proceedings of the 15th International Conference on Web Engineering (ICWE 2015)*, volume 9114 of *LNCS*, pages 515–532. Springer, 2015.

[NKF03] Bashar Nuseibeh, Jeff Kramer, and Anthony Finkelstein. ViewPoints: Meaningful Relationships Are Difficult! In *Proceedings of the 25th International Conference on Software Engineering*, pages 676–681, 2003.

[NRD+16] Petru Nicolaescu, Mario Rosenstengel, Michael Derntl, Ralf Klamma, and Matthias Jarke. View-Based Near Real-Time Collaborative Modeling for Information Systems Engineering. In *Proceedings of the 28th International Conference on Advanced Information Systems Engineering (CAISE 2016)*, volume 9694 of *Lecture Notes in Computer Science*, pages 3–17. Springer International Publishing, 2016.

[NRD+18] Petru Nicolaescu, Mario Rosenstengel, Michael Derntl, Ralf Klamma, and Matthias Jarke. Near real-time collaborative modeling for view-based Web information systems engineering. *Information Systems*, 74:23–39, 2018.

[NRK+14] Petru Nicolaescu, Dominik Renzel, István Koren, Ralf Klamma, Jukka Purma, and Merja Bauters. A Community Information System for Ubiquitous Informal Learning Support. In *Proceedings of IEEE 14th International Conference on Advanced Learning Technologies (ICALT 2014)*, pages 138–140, 2014.

[NTK15] Petru Nicolaescu, Georgios Toubekis, and Ralf Klamma. A Microservice Approach for Near Real-Time Collaborative 3D Objects Annotation on the Web. In *Proceedings of the 14th International Conference on Web-based Learning (ICWL 2015)*, volume 9412 of *Lecture Notes in Computer Science*, pages 187–196. Springer, 2015.

[Obj03] Object Management Group. MDA Guide Version 1.0.1, 2003.

[Obj15] Object Management Group. OMG Unified Modeling Language (OMG UML). [Online] http://www.omg.org/spec/UML/2.5/, 2015.

[Obj17] Object Management Group. Meta-Modeling and the OMG Meta Object Facility (MOF): A White Paper by the OCUP 2 Examination Team. [Online] http://www.omg.org/ocup-2/documents/Meta-ModelingAndtheMOF.pdf, 2017.

[Oli07] Antoni Olivé. *Conceptual Modeling of Information Systems*. Springer-Verlag New York, 2007.

[OMUI06] Gérald Oster, Pascal Molli, Pascal Urso, and Abdessamad Imine. Tombstone Transformation Functions for Ensuring Consistency in Collaborative Editing Systems. In *Proceedings of the 2006 International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 1–10, 2006.

[OO07] Gøran K. Olsen and Jon Oldevik. Scenarios of Traceability in Model to Text Transformations. In *Proceedings of the Third European Conference on Modelling Foundations and Applications*, ECMDA-FA '07, pages 144–156. Springer, 2007.

[Ore17] Jason Orendorff. peeredit on GitHub : *https://github.com/jorendorff/peeredit*, 2017.

[ORK14] Babajide Ogunyomi, Louis M. Rose, and Dimitrios S. Kolovos. On the Use of Signatures for Source Incremental Model-to-text Transformation. In *17th International Conference on Model Driven Engineering Languages and Systems*, MoDELS '14, pages 84–98. Springer International Publishing, 2014.

[OUMI06] Gérald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine. Data Consistency for P2P Collaborative Editing. In *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, CSCW '06, pages 259–268. ACM Press, 2006.

[Pat13] Fabio Paternò. End User Development: Survey of an Emerging Field for Empowering People. *ISRN Software Engineering*, 2013(4):1–11, 2013.

[Pet08] Peter Saint-Andre. XEP-0045: Multi-User Chat, 2008.

[PL14] Maria Paasivaara and Casper Lassenius. Communities of practice in a large distributed agile software development organization – Case Ericsson. *Information and Software Technology*, 56(12):1556–1577, 2014.

[PMSL09] Nuno Preguica, Joan Manuel Marques, Marc Shapiro, and Mihai Letia. A Commutative Replicated Data Type for Cooperative Editing. In *29th IEEE International Conference on Distributed Computing Systems, 2009*, pages 395–403. IEEE, 2009.

[PNH+13] Michael Prilla, Alexander Nolte, Thomas Herrmann, Gwendolyn Kolfschoten, and Stephan Lukosch. Collaborative usage and development of models: State of the art, challenges and opportunities. *International Journal of e-Collaboration*, 9(4):1–16, October 2013.

[Por04] Constance Elise Porter. A Typology of Virtual Communities: A Multi-Disciplinary Foundation for Future Research. *Journal of Computer-Mediated Communication*, 10(1), 2004.

[PR95] Véronique Plihon and Colette Rolland. Modelling ways-of-working. In *Advanced Information Systems Engineering*, pages 126–139, 1995.

[Pul07] Michael Pullin. IMS Content Packaging Research Report, 2007.

[PWD+99] Klaus Pohl, Klaus Weidenhaupt, Ralf Dömges, Peter Haumer, Matthias Jarke, and Ralf Klamma. PRIME: Toward Process-Integrated Modeling Environments. *ACM Transactions on Software Engineering and Methodology*, 8(4):343–410, 1999.

[PZL08] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. RESTful Web Services vs "Big" Web Services: Making the Right Architectural Decision. In *Proceedings of 17th International Conference on World Wide Web (WWW08)*, WWW08, pages 805–814. ACM, 2008.

[RBKJ13] Dominik Renzel, Malte Behrendt, Ralf Klamma, and Matthias Jarke. Requirements Bazaar: Social Requirements Engineering for Community-Driven Innovation. In *21st IEEE International Requirements Engineering Conference: RE 2013*, pages 326–327, 2013.

[RCLK10] Dominik Renzel, Yiwei Cao, Michael Lottko, and Ralf Klamma. Collaborative Video Annotation for Multimedia Sharing between Experts and Amateurs. In *11th International Workshop of the Multimedia Metadata Community on Interoperable Social Multimedia Applications*, volume 583 of *CEUR Workshop Proceedings*, pages 7–14, 2010.

[RCMX06] Balasubramaniam Ramesh, Lan Cao, Kannan Mohan, and Peng Xu. Can distributed software development be agile? *Communications of the ACM*, 49(10):41–46, 2006.

[Ren16] Dominik Renzel. *Information Systems Success Awareness for Professional Long Tail Communities of Practice*. PhD thesis, RWTH Aachen University, Aachen, Germany, 2016.

[Reu04] Reusable eLearning Object Authoring and Delivery. Reload editor introductory manual. [Online] http://www.reload.ac.uk, 2004.

[RJB04] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The*. Pearson Higher Education, 2004.

[RJKL11] Hyun-Gul Roh, Myeongjae Jeon, Jin-Soo Kim, and Joonwon Lee. Replicated Abstract Data Types: Building Blocks for Collaborative Applications. *Journal of Parallel and Distributed Computing*, 71(3):354–368, 2011.

[RKKJ17] Dominik Renzel, Istvan Koren, Ralf Klamma, and Matthias Jarke. Preparing Research Projects for Sustainable Software Engineering in Society. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Society Track (ICSE-SEIS)*, pages 23–32, 2017.

[RKKN15] Dominik Renzel, Ralf Klamma, Miloš Kravčík, and Alexander Nussbaumer. Tracing Self-Regulated Learning in Responsive Open Learning Environments. In *Advances in Web-Based Learning - ICWL 2015*, volume 9412 of *Lecture Notes in Computer Science*, pages 155–164. Springer, 2015.

[RKR+96] Elke Angelika Rundensteiner, Harumi Kuno, Younggook Ra, Viviane Crestana-Taube, Matthew Jones, and Pedro Jose Marron. The MultiView project. *ACM SIGMOD Record*, 25(2):555, 1996.

[RNRG96] Matthias Ressel, Doris Nitsche-Ruhland, and Rul Gunzenhäuser. An Integrating, Transformation-oriented Approach to Concurrency Control and Undo in Group Editors. In *Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work*, CSCW '96, pages 288–297. ACM, 1996.

[RR07] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly, Sebastopol, CA, USA, first edition, 2007.

[RS08] Gustavo Rossi and Daniel Schwabe. Modeling and Implementing Web Applications with Oohdm. In *Web Engineering: Modelling and Implementing Web Applications*, Human-Computer Interaction Series, pages 109–155. Springer-Verlag London Limited, 2008.

[Run92] Elke A. Rundensteiner. MultiView: A Methodology for Supporting Multiple Views in Object-Oriented Databases. In *Proceedings of the 18th VLDB Conference*, pages 187–198. Morgan Kaufmann, 1992.

[Rv07] M. Rosemann and W. M. P. van der Aalst. A configurable reference modelling language. *Inf. Syst.*, 32(1):1–23, March 2007.

[SA09] Peter Saint-Andre. XMPP: lessons learned from ten years of XML messaging. *IEEE Communications Magazine*, 47(4):92–96, 2009.

[SA11] Peter Saint-Andre. Extensible messaging and presence protocol (XMPP): Core, 2011.

[SBL13] Jana Schumann, Tanja Buttler, and Stephan Lukosch. An approach for asynchronous awareness support in collaborative non-linear storytelling. *Computer Supported Cooperative Work (CSCW)*, 22(2):271–308, 2013.

[SC09] David Sun and Chengzheng Sun. Context-Based Operational Transformation in Distributed Collaborative Editing Systems. *IEEE Transactions on Parallel and Distributed Systems*, 20(10):1454–1470, 2009.

[Sch12] Andreas Schmidt. At the boundaries of peer production: The organization of Internet security production in the cases of Estonia 2007 and Conficker. *Telecommunications Policy*, 36(6):451–461, 2012.

[SdM99] Daniel Schwabe, de Almeida Pontes, Rita, and Isbela Moura. OOHDM-Web: An Environment for Implementation of Hypermedia Applications in the WWW. *ACM SIGWEB Newsletter*, 8(2):18–34, 1999.

[SE98] Chengzheng Sun and Clarence Ellis. Operational Transformation in Real-time Group Editors: Issues, Algorithms, and Achievements. In *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work (CSCW '98)*, CSCW '98, pages 59–68. ACM, 1998.

[Seg07a] Judith Segal. End-User Software Engineering and Professional End-User Developers. In *Dagstuhl Seminar Proceedings*, volume 07081 of *End-User Software Engineering*, pages 1–2. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2007.

[Seg07b] Judith Segal. Some Problems of Professional End User Developers. In *IEEE Symposium on Visual Languages and Human-Centric Computing, 2007*, pages 111–118. IEEE Computer Soc, 2007.

[SG89] Susan Leigh Star and James R. Griesemer. Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39. *Social Studies of Science*, 19(3):387–420, 1989.

[Sha07] Nalin Sharda. Applying Movement Oriented Design to Create Educational Stories. *International journal of Learning*, 13(12):177–184, 2007.

[Sir14] Sirius - The easiest way to get your own modeling tool: Graphical Editors for your DSL, 2014.

[SJYZ97] Chengzheng Sun, Xiaohua Jia, Yun Yang, and Yanchun Zhang. REDUCE: a prototypical cooperative editing system. In *HCI (1)*, pages 89–92, 1997.

[SJZ+98] Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Achieving Convergence, Causality Preservation, and Intention Preservation in Real-time Cooperative Editing Systems. *ACM Transactions on Computer-Human Interaction*, 5(1):63–108, 1998.

[SK06] Wieland Schwinger and Nora Koch. Modeling Web Applications. In *Web Engineering - Systematic Development of Web Applications*, pages 39–64. John Wiley and Sons Ltd, 2006.

[SKJ03] Marc Spaniol, Ralf Klamma, and Matthias Jarke. ATLAS: A web-Based Software Architecture for Multimedia E-Learning Environments in Virtual Communities. In *Proceedings of the 2nd International Conference on Advances in Web-based Learning (ICWL 2003)*, volume 2783 of *Lecture Notes in Computer Science*, pages 193–205. Springer, 2003.

[SKJR06] Marc Spaniol, Ralf Klamma, Holger Janßen, and Dominik Renzel. LAS: A Lightweight Application Server for MPEG-7 Services in Community Engines. In *Proceedings of I-KNOW '06, 6th International Conference on Knowledge Management*, J.UCS (Journal of Universal Computer Science) Proceedings, pages 592–599. Springer-Verlag, 2006.

[SKSJ06] Marc Spaniol, Ralf Klamma, Nalin Sharda, and Matthias Jarke. Web-Based Learning with Non-linear Multimedia Stories. In *Advances in Web-based Learning: Revised Papers*, volume 4181 of *Lecture Notes in Computer Science*, pages 249–263. Springer, 2006.

[SKW97] Oliver Stiemerling, Helge Kahler, and Volker Wulf. How to make software softer—designing tailorable applications. In *Proceedings of the 2nd Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, pages 365–376, 1997.

[SL07] Louise Stoll and Karen Seashore Louis. *Professional learning communities: Divergence depth and dilemmas*. Professional learning. McGraw-Hill Open Univ. Press, Maidenhead, 2007.

[SLH14] Oliver Schmid, Agnes Lisowska Masson, and Béat Hirsbrunner. Real-time collaboration through web applications: an introduction to the Toolkit for Web-based Interactive Collaborative Environments (TWICE). *Personal and Ubiquitous Computing*, 18(5):1201–1211, 2014.

[SMB95] Mark S. Silver, M. Lynne Markus, and Cynthia Mathis Beath. The Information Technology Interaction Model: A Foundation for the MBA Core Course. *MIS Quarterly*, 19(3):361, 1995.

[SPBZ11a] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-Free Replicated Data Types. In *Stabilization, Safety, and Security of Distributed Systems*, volume 6976 of *Lecture Notes in Computer Science*, pages 386–400. Springer Berlin Heidelberg, 2011.

[SPBZ11b] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types, 2011.

[SR98] Daniel Schwabe and Gustavo Rossi. An Object Oriented Approach to Web-based Applications Design. *TAPOS*, 4(4):207–225, 1998.

[SS06] David Sun and Chengzheng Sun. Operation Context and Context-based Operational Transformation. In *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, CSCW '06, pages 279–288. ACM Press, 2006.

[Sun02] Chengzheng Sun. Undo As Concurrent Inverse in Group Editors. *ACM Transactions on Computer-Human Interaction*, 9(4):309–361, 2002.

[Sun12] Sun, Chengzheng. OT FAQ: Operational Transformation Frequently Asked Questions and Answers. [Online] http://www3.ntu.edu.sg/home/czsun/projects/otfaq/, 13.06.2012.

[SVP05] Mohanbir Sawhney, Gianmario Verona, and Emanuela Prandelli. Collaborating to Create: The Internet as a Platform for Customer Engagement in Product Innovation. *Journal of Interactive Marketing*, 19(4):4–17, 2005.

[SWK06] Andrea Schauerhuber, Manuel Wimmer, and Elisabeth Kapsammer. Bridging Existing Web Modeling Languages to Model-driven Engineering: a Metamodel for WebML. In *Workshop Proceedings of the Sixth International Conference on Web Engineering*, page 5, 2006.

[SXA14] Chengzheng Sun, Yi Xu, and Agustina Agustina. Exhaustive Search of Puzzles in Operational Transformation. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 519–529. ACM, 2014.

[SXN17] Chengzheng Sun, Yi Xu, and Agustina Ng. Exhaustive Search and Resolution of Puzzles in OT Systems Supporting String-Wise Operations. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, pages 2504–2517, 2017.

[SXS+06] Chengzheng Sun, Steven Xia, David Sun, David Chen, Haifeng Shen, and Wentong Cai. Transparent Adaptation of Single-User Applications for Multi-User Real-Time Collaboration. *ACM Transactions on Computer-Human Interaction*, 13(4):531–582, 2006.

[Tar17] Dominic Tarr. Commutative Replicated Data Types: *https://github.com/dominictarr/crdt*, 2017.

[TEN10] TEN Competence Foundation. ReCourse Learning Design Editor, 2010.

[TH77] Daniel Teichroew and Ernest Hershey. PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems. *IEEE Transactions on Software Engineering*, 3(1):41–48, 1977.

[Tig09] Tigris.org. ArgoUML, 2009.

[Tim05] O'Reilly Tim. What Is Web 2.0? Design Patterns and Business Models for the Next Generation of Software. 2005.

[TL02] James C. Toole and Karen Seashore Louis. The Role of Professional Learning Communities In International Education. In *Second International Handbook of Educational Leadership and Administration*, pages 245–279. Springer Netherlands, 2002.

[TS03] Richard H. Thaler and Cass R. Sunstein. Libertarian Paternalism. *American Economic Review*, pages 175–179, 2003.

[TS08] Richard H. Thaler and Cass R. Sunstein. *Nudge: Improving Decisions About Health, Wealth, and Happiness*. Concentrated knowledge for the busy executive. Yale University Press, New Haven, CT, USA, 2008.

[TW08] Don Tapscott and Anthony D. Williams. *Wikinomics: How Mass Collaboration Changes Everything*. Portfolio, USA, 2008.

[VBJM14] Juan Manuel Vara, Veronica A. Bollati, Alvaro Jimenez, and Esperanza Marcos. Dealing with Traceability in the MDD of Model Transformations. *IEEE Transactions on Software Engineering*, 40(6):555–583, 2014.

[VMP14] Vladimir Viyovic, Mirjam Maksimovic, and Branko Perisic. Sirius: A Rapid Development of DSM Graphical Editor. In *Proceedings of the 18th International Conference on Intelligent Engineering Systems (INES)*, pages 233–238. IEEE, 2014.

[WBP+05] Wim Westera, Francis Brouns, Kees Pannekeet, Jose Janssen, and Jocelyn Manderveld. Achieving E-learning with IMS Learning Design - Workflow Implications at the Open University of the Netherlands. *Educational Technology and Society*, 8(3):216–225, 2005.

[Wen98] Etienne Wenger. *Communities of Practice: Learning, Meaning, and Identity*. Learning in doing. Cambridge University Press, Cambridge, UK, 1998.

[Wen00] Etienne Wenger. Communities of Practice and Social Learning Systems. *Organization*, 7(2):225–246, 2000.

[Wen10] Etienne Wenger. Communities of Practice and Social Learning Systems: the Career of a Concept. In *Social Learning Systems and Communities of Practice*, pages 179–198. Springer London, 2010.

[Wil08] Dennis M. Wilkinson. Strong Regularities in Online Peer Production. In *Proceedings of the 9th ACM Conference on Electronic Commerce - EC '08*, pages 302–309. ACM Press, 2008.

[WJ04] Volker Wulf and Matthias Jarke. The Economics of End-User Development. *Communications of the ACM*, 47(9):41–42, 2004.

[WMS02] Etienne Wenger, Richard McDermott, and William M. Snyder. *Cultivating Communities of Practice*. Harvard University Press, Boston, MA, USA, 2002.

[WTWT15] Etienne Wenger-Trayner and Beverly Wenger-Trayner. Introduction to Communities of Practice, 2015.

[WUM09] Stéphane Weiss, Pascal Urso, and Pascal Molli. Logoot: A Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks. In *29th IEEE International Conference on Distributed Computing Systems, 2009*, pages 404–412. IEEE, 2009.

[WWS09] Etienne Wenger, Nancy White, and John D. Smith. *Digital Habitats: Stewarding Technology for Communities*. CPSquare, Portland, OR, USA, 1st edition, 2009.

[XS16] Yi Xu and Chengzheng Sun. Conditions and Patterns for Achieving Convergence in OT-Based Co-Editors. *IEEE Transactions on Parallel and Distributed Systems*, 27(3):695–709, 2016.

[XSL14] Yi Xu, Chengzheng Sun, and Mo Li. Achieving Convergence in Operational Transformation: Conditions, Mechanisms and Systems. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 505–518. ACM, 2014.

[XZS00] Liyin Xue, Kang Zhang, and Chengzheng Sun. Conflict Control Locking in Distributed Cooperative Graphics Editors. In *Proceedings of the 1st International Conference on Web Information Systems Engineering*, pages 401–408, 2000.

[YF07] Yunwen Ye and Gerhard Fischer. Designing for Participation in Sociotechnical Software Systems. In *Universal Acess in Human Computer Interaction. Coping with Diversity*, volume 4554 of *Lecture Notes in Computer Science*, pages 312–321. Springer Berlin Heidelberg, 2007.

[Yu95] Eric Yu. From organization models to system requirements: a 'cooperating agents' approach. In *Cooperative Information Systems*, 1995.

[Yu97] Eric Yu. Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In *Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering (RE'97)*, pages 226–235, 1997.

[Zac03] John A. Zachman. *The Zachman Framework: A Primer for Enterprise Engineering and Manufacturing*. Zachman Framework Associates, 2003.

List of Figures

List of Tables

Appendix A

List of Abbreviations

Abbreviation	Description	First mentioned
ANT	Actor Network Theory	p. 22
API	Application Programmable Interface	p. 1
ATLAS	Architecture for Transcription, Localization and Addressing Systems	p. 21
BME	Begin Middle End	p. 189
CAE	Community Application Editor	p. 80
CIS	Community Information System	p. 15
CMS	Content Management System	p. 1
CoP	Communities of Practice	p. 1
CRDT	Conflict-free Replicated Data Type	p. 46
CVG	Closed-View Generation	p. 130
DOM	Document Object Model	p. 95
DSL	Domain Specific Language	p. 34
EUD	End-User Development	p. 2
GLS	Guidance Model Specification	p. 138
GML	Guidance Modeling Language	p. 138
HTML	Hypertext Markup Language	p. 7
HTTP	Hypertext Transfer Protocol	p. 143
IDC	International Data Corporation	p. 5
IEEE	Institute of Electrical and Electronics Engineers	p. 37
IFML	Interaction Flow Modeling Language	p. 31
IMS LD	IMS Learning Design	p. 4

IWC	Inter-widget Communication	p.	71
LAS	Lightweight Application Server	p.	50
MDA	Model-Driven Architecture	p.	28
MDWE	Model-Driven Web Engineering	p.	3
MIST	Media Integrated Storytelling	p.	189
MOD	Movement Oriented Design	p.	187
MPEG-7	Multimedia Content Description Interface	p.	50
PLE	Personal Learning Environment	p.	53
NRT	Near Real-Time	p.	95
P2P	Peer-to-Peer	p.	1
PLE	Personal Learning Environment	p.	53
REST	Representational State Transfer	p.	24
ROLE	Responsive Open Learning Environments	p.	10
RSS	Rich Site Summary	p.	24
RTCES	Real-Time Collaborative Editing System	p.	41
SOA	Service Oriented Architecture	p.	23
SME	Small and Medium Enterprise	p.	54
SOAP	Simple Object Access Protocol	p.	24
SVG	Scalable Vector Graphics	p.	45
OT	Operational Transformation	p.	43
VC	Virtual Campfire	p.	50
VLS	Visual Language Specification	p.	130
VML	Visual Modeling Language	p.	130
W3C	World Wide Web Consortium	p.	69
WebRTC	Web Real-Time Communication	p.	25
XEP	XMPP Extension Protocol	p.	120
XML	Extensible Markup Language	p.	28
XMPP	Extensible Messaging and Presence Protocol	p.	25

Appendix B

Own Publications

Relevant Refereed Publications

[DEN+14] Michael Derntl, Stephan Erdtmann, Petru Nicolaescu, Ralf Klamma, and Matthias Jarke. Echtzeitmetamodellierung im Web-Browser. In *Proceedings of Modellierung 2014*, pages 65–80. Gesellschaft fur Informatik e.V., 2014.

[dND+16] Peter de Lange, Petru Nicolaescu, Michael Derntl, Matthias Jarke, and Ralf Klamma. Community Application Editor: Collaborative Near Real-Time Modeling and Composition of Microservice-based Web Applications. In *Proceedings of the Modellierung 2016 Workshopband*, pages 123–127, 2016.

[DNE+15] Michael Derntl, Petru Nicolaescu, Stephan Erdtmann, Ralf Klamma, and Matthias Jarke. Near Real-Time Collaborative Conceptual Modeling on the Web. In *Proceedings of the 34th International Conference on Conceptual Modeling (ER 2015)*, volume 9381 of *Lecture Notes in Computer Science*, pages 344–357. Springer International Publishing, 2015.

[dNKK16] Peter de Lange, Petru Nicolaescu, Ralf Klamma, and István Koren. DevOpsUse for Rapid Training of Agile Practices Within Undergraduate and Startup Communities. In *Proceedings of the 11th European Conference on Technology Enhanced Learning (EC-TEL 2016)*, volume 9891 of *Lecture Notes in Computer Science*, pages 570–574. Springer International Publishing, 2016.

[DNTK13] Michael Derntl, Petru Nicolaescu, Bezunesh Alemu Terkik, and Ralf Klamma. SynC-LD: Synchronous Collaborative IMS Learning Design Authoring on the Web. In *Proceedings of the 8th European Conference on Technology Enhanced Learning (EC-TEL 2013)*, volume 8095 of *LNCS*, pages 540–543. Springer-Verlag, 2013.

[dPKJ17] Peter de Lange, Nicolaescu Petru, Ralf Klamma, and Matthias Jarke. Engineering Web Applications Using Real-Time Collaborative Modeling. In *Proceedings of the 23rd International Conference on Collaboration and Technology (CRIWG 2017)*, volume 10391 of *Lecture Notes in Computer Science*, pages 213–228, 2017.

[KNK] Miloš Kravčík, Petru Nicolaescu, and Ralf Klamma. Informal Learning at the Workplace via Adaptive Video. In *Proceedings of the UMAP, ProS: Workshop on UMAP Projects Synergy*, volume 1181, pages 35–38.

[KNK14] Dejan Kovachev, Petru Nicolaescu, and Ralf Klamma. Mobile Real-Time Collaboration for Semantic Multimedia. *Mobile Networks and Applications*, 19(5):635–648, 2014.

[KNK15] István Koren, Petru Nicolaescu, and Ralf Klamma. Collaborative Drawing Annotations on Web Videos. In *Proceedings of the 15th International Conference on Web Engineering (ICWE 2015)*, volume 9114 of *LNCS*, pages 671–674. Springer, 2015.

[KNSK17] Miloš Kravčík, Petru Nicolaescu, Aarij Siddiqui, and Ralf Klamma. Adaptive Video Techniques for Informal Learning Support in Workplace Environments. In *Proceedings of the First International Symposium in Emerging Technologies for Education*, volume 10108 of *Lecture Notes in Computer Science*, pages 533–543, 2017.

[NDK13] Petru Nicolaescu, Michael Derntl, and Ralf Klamma. Browser-based Collaborative Modeling in Near Real-Time. In *Proceedings of the 9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom 2013)*, pages 335–344, 2013.

[NJDK15] Petru Nicolaescu, Kevin Jahns, Michael Derntl, and Ralf Klamma. Yjs: A Framework for Near Real-Time P2P Shared Editing on Arbitrary Data Types. In *Proceedings of the 15th International Conference on Web Engineering (ICWE 2015):* ***Best demo award***, volume 9114 of *LNCS*, pages 675–678. Springer, 2015.

[NJDK16] Petru Nicolaescu, Kevin Jahns, Michael Derntl, and Ralf Klamma. Near Real-Time Peer-to-Peer Shared Editing on Extensible Data Types. In *Proceedings of*

the 19th International Conference on Supporting Group Work (GROUP 2016). ACM, 2016.

[NK14] Petru Nicolaescu and Ralf Klamma. SeViAnno 2.0: Web-Enabled Collaborative Semantic Video Annotation Beyond the Obvious. In *Proceedings of the 12th International Workshop on Content-Based Multimedia Indexing (CMBI 2014)*, pages 1–6, 2014.

[NK15] Petru Nicolaescu and Ralf Klamma. A Methodology and Tool Support for Widget-based Web Application Development. In *Proceedings of the 15th International Conference on Web Engineering (ICWE 2015)*, volume 9114 of *LNCS*, pages 515–532. Springer, 2015.

[NRD+16] Petru Nicolaescu, Mario Rosenstengel, Michael Derntl, Ralf Klamma, and Matthias Jarke. View-Based Near Real-Time Collaborative Modeling for Information Systems Engineering. In *Proceedings of the 28th International ConferenceAdvanced Information Systems Engineering (CAISE 2016):* ***Best paper award; ACM Computing Reviews Best of Computing 2016 list***, volume 9694 of *Lecture Notes in Computer Science*, pages 3–17. Springer International Publishing, 2016.

[NRD+17] Petru Nicolaescu, Mario Rosenstengel, Michael Derntl, Ralf Klamma, and Matthias Jarke. Near real-time collaborative modeling for view-based Web information systems engineering. *Information Systems*, to appear, 2017.

[NRK+14] Petru Nicolaescu, Dominik Renzel, István Koren, Ralf Klamma, Jukka Purma, and Merja Bauters. A Community Information System for Ubiquitous Informal Learning Support. In *Proceedings of the 14th IEEE International Conference on Advanced Learning Technologies (ICALT 2014)*, pages 138–140, 2014.

[NTK15] Petru Nicolaescu, Georgios Toubekis, and Ralf Klamma. A Microservice Approach for Near Real-Time Collaborative 3D Objects Annotation on the Web. In *Proceedings of the 14th International Conference on Web-Based Learning (ICWL 2015)*, volume 9412 of *Lecture Notes in Computer Science*, pages 187–196. Springer, 2015.

Relevant Non Refereed Publications

[DHK+14] Michael Derntl, Anna Hannemann, Ralf Klamma, István Koren, Petru Nicolaescu, Dominik Renzel, Miloš Kravčík, Mohsen Shahriari, Jukka Purma, Martin Bachl, Edward Bellamy, Raymond Elferink, Vladimir Tomberg, Dieter Theiler, and Patricia Santos. Customizable Architecture for Flexible Small-Scale Deployment, 2014.

[DKK+13] Michael Derntl, Ralf Klamma, István Koren, Miloš Kravčík, Petru Nicolaescu, Dominik Renzel, Kiarii Ngua, Jukka Purma, Graham Attwell, Owen Gray, Tobias Ley, Vladimir Tomberg, Christina Henry, Chris Whitehead, Dieter Theiler, Christoph Trattner, Ronald K. Maier, Markus Manhart, Maria Schett, and Stefan Thalmann. Initial Architecture for Fast Small-Scale Deployment: D6.1, 2013.

[KKK+16] Ralf Klamma, Zinayida Kensche, Miloš Kravčík, Dominik Renzel, István Koren, Peter de Lange, Kateryna Neulinger, Petru Nicolaescu, Mohsen Shahriari, and Georgios Toubekis. Advanced Community Information Systems (ACIS) - Annual Report 2016. Aachen, Germany, 2016.

[KKN+15] Ralf Klamma, István Koren, Petru Nicolaescu, Dominik Renzel, Miloš Kravčík, Mohsen Shahriari, Michael Derntl, Gilbert Peffer, and Raymond Elferink. DevOpsUse - Scaling Continuous Innovation: Project Deliverable, 2015.

Other Publications

[DKN+14] Michael Derntl, István Koren, Petru Nicolaescu, Dominik Renzel, and Ralf Klamma. Blueprint for Software Engineering in Technology Enhanced Learning Projects. In *Proceedings of the 9th European Conference on Technology*

Enhanced Learning (EC-TEL 2014), volume 8719 of *LNCS*, pages 404–409. Springer Switzerland, 2014.

[dNK17] Peter de Lange, Petru Nicolaescu, and Ralf Klamma. VIRTUS virtual VET centre (V3C): A learning platform for virtual vocational education and training. In *Proceedings of the 12th European Conference on Technology Enhanced Learning, (EC-TEL 2017)*, pages 500–503, 2017.

[DRN+15] Michael Derntl, Dominik Renzel, Petru Nicolaescu, István Koren, and Ralf Klamma. Distributed Software Engineering in Collaborative Research Projects. In *Proceedings of the 10th IEEE International Conference on Global Software Engineering (ICGSE 2015)*, pages 105–109. IEEE Computer Society, 2015.

[KRN+14] Dejan Kovachev, Dominik Renzel, Petru Nicolaescu, István Koren, and Ralf Klamma. DireWolf: A Framework for Widget-based Distributed User Interfaces. *Journal of Web Engineering*, 13(3&4):203–222, 2014.

[KRNK13] Dejan Kovachev, Dominik Renzel, Petru Nicolaescu, and Ralf Klamma. DireWolf - Distributing and Migrating User Interfaces for Widget-Based Web Applications. In *Proceedings of the 13th International Conference on Web Engineering (ICWE 2013)*, volume 7977 of *LNCS*, pages 99–113. Springer, 2013.

Appendix C

Curriculum Vitae

Personal information	
First name/Surname	**Petru NICOLAESCU**
Address	Ahornstr. 55 52074 Aachen, Germany
Telephones	Fix : +49 241 80 21516
E-mail	nicolaescu@dbis.rwth-aachen.de
Nationality	Romanian
Date/Place of birth	06th of August 1986, Bucharest, Romania
Web	http://dbis.rwth-aachen.de/cms/staff/nicolaescu

Education	
05.2012 - present	**PhD Candidate** at Faculty of Mathematics, Computer Science and Natural Sciences, Computer Science Department, **RWTH Aachen University**, Germany
10.2009 - 02.2012	**Software Systems Engineering Master of Science** at Faculty of Mathematics, Computer Science and Natural Sciences, **RWTH Aachen University**, Germany
10.2005 - 07.2009	**Bachelor in Computer Science** at Faculty Of Engineering in Foreign Languages, English Section, **POLITEHNICA University of Bucharest**
Professional Experience	
05.2012 - present	**Research and Teaching Assistant** at **RWTH Aachen University,** Advanced Community Information Systems Group (ACIS) at the Chair of Information Systems and Databases (Prof. Matthias Jarke)
11.2009 - 11.2011	**Student Researcher** (HiWi) at the **Chair for Information Systems and Databases**, RWTH Aachen
11.2007 - 09.2009	**Software Developer** (part-time) at **Siveco Romania SA**, E-Health department

Aachen, 19.09.2017

Petru Nicolaescu