Model-based Real-time Synchronization

Stephan Krusche
Department of Computer Science,
Technische Universitaet Muenchen
krusche@in.tum.de

Bernd Bruegge
Department of Computer Science,
Technische Universitaet Muenchen
bruegge@in.tum.de

ABSTRACT

In this paper we describe an approach for model-based real-time synchronization. We present an extension of the EMFStore platform which allows multiple collaborators to connect to each other directly via peer-to-peer and to synchronize changes on model instances with each other in real-time. With this approach we allow users to collaboratively work literally on the same model instance. We argue that this approach avoids serious conflicts and reduces the problem of outdated model instances.

Keywords

Synchronization, Collaboration, Same-Time, Real-Time, Versioning, Models, Conflict Avoidance, Face-to-face, Peer-to-Peer

1. INTRODUCTION

Object-oriented programming allows software developers to decouple the different subsystems of the software to be developed. With the Model-View-Controller architectural style a software developer divides the system into entity objects (Model), control objects (Controller) and boundary objects (View) [Bus99]. If developers want to persist data over the application life-cycle, they need to store representations of the entity objects in the file system, in a database or in the cloud.

Applications, that support multiple users to collaboratively create, edit and share data, need to be able to synchronize these entity objects between each other when changes on existing objects occur, new objects are created or existing ones are deleted. Developers can e.g. achieve this by using client-server or peer-to-peer architectures. However, persisting data and keeping it consistent and up-to-date can be quite difficult, especially with many entity objects structured in a complex data model.

Koegel and Helming developed the EMFStore [HK13], which acts as a data model repository for EMF[1] models and which is capable to synchronize arbitrary[2] data model instances between multiple clients. It allows semantic versioning of models and supports conflict detection, merging and branching. The architecture of the EMFStore is shown in fig. 1.

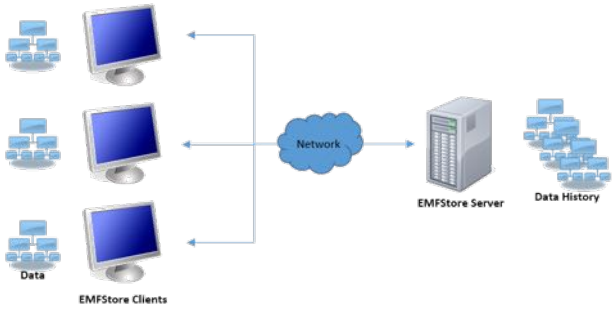

Figure 1: Architecture of the EMFStore [HK13]

The EMFStore uses operation-based change tracking as shown in fig. 2. An operation is either an atomic change of the model instance or a composite of multiple atomic changes (composite operation). [KHS09]

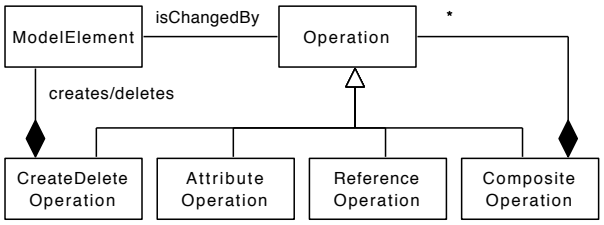

Figure 2: Taxonomy of operations [KHS09]

The EMFStore allows users to checkout a model instance, automatically tracks changes on the instance in an operation-based way, and allows users to commit these changes back

[1]Eclipse Modeling Framework, see `http://www.eclipse.org/modeling/emf`.

[2]The term "arbitrary model instances" refers to the fact, that there is no limitation regarding the types of references or regarding cycles. References to abstract classes or cycles within the object graph can be difficult to synchronize. When using the EMFStore platform, the developer only needs to follow the EMF standard and such cases are handled automatically.

to the repository on the server. If the local version of the model instance is outdated, the user needs to update to the newest version first, before he can commit. Additionally a user is able to create branches of a model instance.

The EMFStore is comparable to Subversion (SVN)[3], a version control system for textual artifacts because it acts as a single remote repository. [KH10] The checkout-update-commit approach allows users to collaborate asynchronously on model instances. The EMFStore has advantages compared to the synchronization of textual model representations (e.g. in XML) using a text-based version control system like SVN which works on a line-oriented level. Models are not managed on a line-oriented level, thus merging them with SVN could lead to unnecessary conflicts because of the impedance mismatch described by Nguyen et. al in [NMBT05].

The EMFStore improves the conflict detection and the merge of two model instances because it uses an appropriate level of abstraction. Changes on model instances are recorded as operations on objects and conflicts can be detected on a finer level. However the EMFStore currently does not allow users to collaborate synchronously. This means the EMFStore covers the right side of the CSCW[4] matrix shown in fig. 3 adapted from [Joh88].

	same time (synchronous)	**different time** (asynchronous)
same place (colocated)	Face to face interactions	Continuous task
different place (remote)	Remote interactions	Communication & coordination

Figure 3: Computer supported cooperative work matrix (adapted from [Joh88])

In this paper we extend Koegel's and Heliming's approach with the EMFStore to allow users to collaboratively work on model instances in real-time. With our approach we also cover the left side (same-time) of the CSCW matrix in fig. 3.

2. EXTENSION OF THE EMFSTORE APPROACH

We develop a peer-to-peer (P2P) component for the EMFStore platform that allows to synchronize changes on models between different collaborators in real-time. To achieve this we implement additional subsystems into a P2P framework component that allows clients to communicate directly to each other, thus becoming peers. This framework component is easy to use and a developer can simply plug it into an existing ECP[5] application. The central repository is still available to achieve global persistence, but peers prefer to exchange model instances and changes on these model instance directly with each other to reduce the exchange time.

[3] `http://subversion.apache.org`
[4] Computer supported cooperative work
[5] Eclipse Client Platform, see `http://www.eclipse.org/ecp`

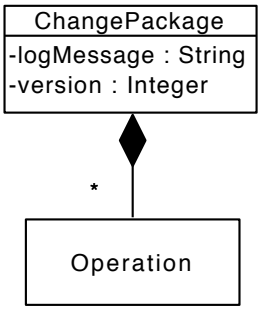

Figure 4: Change package in the current EMFStore platform

In the current implementation of the EMFStore[6] a change package always corresponds to one commit. The version number is increased for each commit. Like in SVN, a user can only commit to the remote repository[7], if he has the most recent version of a branch (or the trunk). If multiple branches are used, a new commit always gets the last version number increased by one. This leads e.g. to a branch situation as shown in fig. 5.

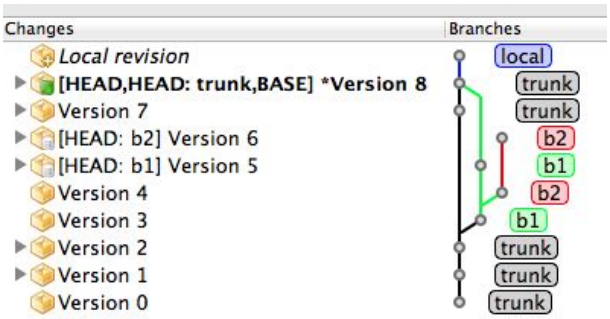

Figure 5: Branches in EMFStore

To realize real-time synchronization, our extension needs to send change packages immediately to other peers, bypassing the server. If we would simply increase the version number of change packages on peers, this could lead to unwanted situations because the change packages are not synchronized with the server. Two different peers could have the same version number, but different versions of the model instance. Thus we need to use a different approach in our extension.

Based on the implementation of git [Tor13], we use unique hash values to identify change packages (comparable to git commits) and save the parent identifier of a change package to know the right order of the change packages. An example of a git repository structure is shown in fig. 6 [Cha09]. In the example there are three commits (green) and each of them points to the previous commit. Each commit also points to a working directory called tree (blue) which includes files and text lines within these files (red).

To achieve this structure, we need to adapt the change package class in our extension as shown in fig. 7. We change

[6] The current version of the EMFStore was 1.1.0 when writing this paper.
[7] Local commits like e.g. in git are currently not possible with the EMFStore.

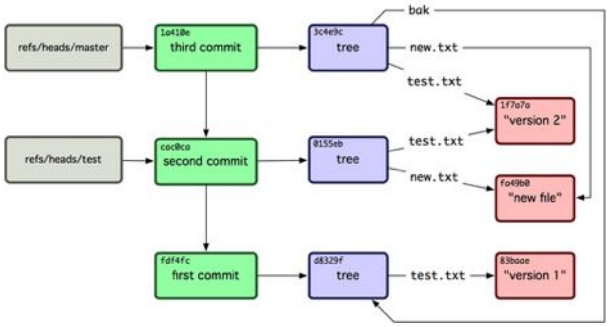

Figure 6: Objects in a git repository [Cha09]

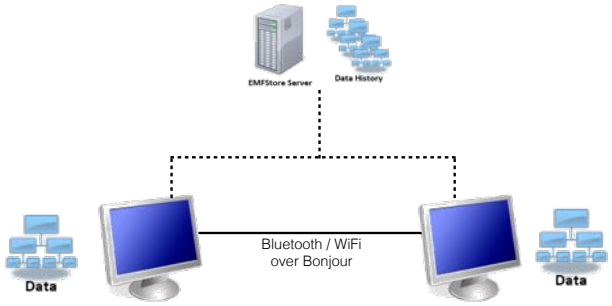

Figure 8: Colocated model-based real-time synchronization via Bonjour

the version to a string attribute in order to store unique hash values. This solves the problem of conflicting version numbers in the case when multiple clients increase their version number locally. We additionally insert a relationship from a change package to its parent change package like the pointer to previous commits in git. While this adaption breaks the compatibility of our extension with existing releases of the EMFStore, it is necessary to prevent the above described unwanted situations where different versions of a model instance have the same version number.

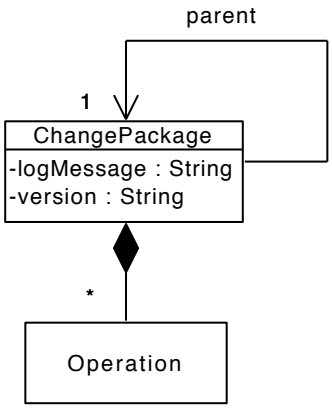

Figure 7: Adapted change package for real-time synchronization

With this adaption the peer-to-peer extension can synchronize change packages between multiple peers directly and is still able to synchronize these change packages with remote repositories. It might happen that multiple peers want to push a new change package to the remote repository with the same parent. In this case the peer wins which pushes the changes first (in a timed order). The second peer first needs to pull the change package from the server and needs to perform a merge or rebase operation. [Swi08] We also gain further possibilities like e.g. cherry picking certain change packages and reverting them. Additionally it would be possible to reorder change packages or to combine the operations of multiple change packages into one change package.

We use the same approach for change-based operation tracking as described in [KHS09], thus we are able to exchange any object-oriented model instance providing a very generic way of exchanging data in real-time.

Depending on the situation we allow to use different technologies for real-time synchronization. In colocated situations peers can connect via Bonjour[8], Apple's implementation of zero-configuration networking. Bonjour includes service discovery, address assignment and hostname resolution. Peers connect with each other using Bonjour in local area networks (e.g. via WiFi) or via Bluetooth as shown in fig. 8. This approach allows peers to exchange data without a connection to the EMFStore server, e.g. in mobile situations without internet connection and covers the same time and same place entry of the CSCW matrix in fig. 3.

If two peers are not in the same place and thus not connected in a local area network, they can also connect remotely over the Internet. In such situations peers cannot connect directly to each other, if port forwarding is disabled in the network configuration which is the case in most local area network configurations. Therefore we need a server in this situation to make peers available to each other, to keep a connection between multiple peers and to transport the data between the peers. However such a server does not need to have complex behavior, it just needs to mimic the behavior of a Bonjour router.

Traditionally developers need to workaround the request-response principle of a client-server connection by using approaches like long-polling[9] to enable the server to send data to the client. In 2011, the WebSocket protocol[10] was standardized as a way to create a connection between client and server over HTTP and to provide full-duplex communications channels via TCP sockets. Our peer-to-peer approach over the Internet uses a WebSocket server which forwards changes from one collaborator of the session to all other collaborators as shown in fig. 9.

We do not require hard real-time, but soft real-time as described in [Kop11]. This means, that the synchronized changes are still valid even if they are received in a delay of a few seconds.

Users who want to collaborate, start a real-time collaboration session. They need to connect to each other and synchronize their model instances before starting to collaborate. To synchronize, they can checkout model instances which are not available yet when the other user allows the

[8] `https://www.apple.com/support/bonjour`

[9] Long polling is an approach to overcome the limit that a client always needs to start a request, before the server can send data to the client. Such workarounds are collected under the umbrella term "Comet", see e.g. [MC08].

[10] `http://www.websocket.org`

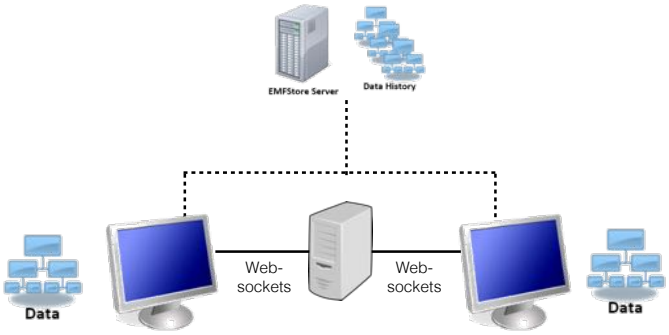

Figure 9: Remote model-based real-time synchronization via Websockets

checkout. If all collaborators have the same model instance, but not in the same version, they need to synchronize to the newest (or a specific) version of the model instance.

Additionally a user can specify to include the EMFStore repository as a peer so that all changes are immediately synchronized with it. This can be compared to Dropbox where all changes on a file are immediately synchronized to the server and all connected users see the change immediately. This prevents users from working on outdated versions of their model instance, because at startup the application immediately asks the EMFStore server (and all other possible collaborators) to synchronize and to update to the newest version.

Koegel et al. describe the idea of collaborative model merging in [KNHH10]. The EMFStore platform is able to detect conflicts and makes them part of the communication model. This allows conflict visualization and conflict resolution, e.g. with visual merging, after a conflict occurs. While this is still a functioning approach in our extension, we particularly focus on a different approach, conflict avoidance. We motivate users to work collaboratively from the beginning and synchronize changes immediately when they occur. If the synchronization is fast enough, i.e. below a few seconds, the users can avoid conflicts with this behavior. Even if conflicts might occur in situations with a higher network delay, they are still small and simple to resolve. With this behavior, users do not work on outdated model instances and thus are always up-to-date.

To further prevent conflicts, we implement privileged access through locking as described in [GM94]. If the user is currently editing a certain model element, the application can send a lock message for this object (and potentially all contained objects) over the P2P component of the framework. Other collaborators can then see that the model element is currently edited by a different user. While this locking mechanism can avoid conflicts, it is also an impulse for further communication between the collaborators. The approach is only a technical implementation to make logical conflicts visible. If a user e.g. wants to delete an object whereas another user wants to change an attribute of this object, there is still a need for negotiation between both collaborators. The realization of this negotiation is currently out of scope in our extension. However in colocated situations collaborators can directly talk to each other, and in remote situations collaborators can use existing audio and video based online tools to communicate, e.g. Skype, to even allow face-to-face discussions.

If the application does not make use of the locking mechanism and a conflict occurs, the framework notifies the application which can then ask the user for a solution of the conflict, e.g. through visual merging.

3. RELATED WORK

In the area of collaborative document creation, different tools already allow real-time collaboration. Google Drive, Apple's iWork and Microsoft Office 365 are prominent tools which enable the creation and editing of documents at the same time over the internet. These tools already cover all four areas of CSCW. However they do not use modern techniques like WebSockets or Bonjour and are not implemented in a model-based way.

Google Drive provides a "Realtime API" for external applications[11]. A developer could use this API to implement a real-time application. However this API itself is limited to document-based synchronization, such as lists, strings and key-value maps. It does not specifically support model-based synchronization and complex object graphs. The developer would need to implement the details of the synchronization of model instances and need to handle complex situations like relations to abstract classes or cycles in the object graph on his own. Google Drive uses a web application model in which a long-held HTTP request allows the web service to realize push notifications, also referred to by the term Comet, compare e.g. [MC08]. Even if such a mechanism realizes real-time communication, it is hard to implement and maintain it with complex object graphs. Another problem of Google Drive is the need to use external services by Google, which might not be allowed due to security concerns.

Dropbox[12] is a service that allows users to synchronize data between multiple devices. It allows sharing of files between multiple users. Dropbox uses some type of peer-to-peer protocol: If users are in the same local area network, Dropbox synchronizes document with all users in the network as well as with the cloud. While it recognizes synchronization conflicts, it does not resolve them, even if it would be possible without user intervention (e.g. changes in non-overlapping areas of a text document). However it also does not support model-based synchronization and relies on external servers.

Firebase[13] is a service that promises to alleviate the creation of "Realtime Apps". It offers a "The Realtime Application Platform" with a REST[14] API and a NoSQL[15] data store. However Firebase does not support arbitrary complex object graphs and is not able to synchronize cyclic dependencies in object graphs out of the box.

4. CONCLUSION

We introduced a peer-to-peer extension for the EMFStore platform which allows to synchronize changes on arbitrary model instances instantly with all collaborators of a real-time collaboration session. Our peer-to-peer approach is a mimicry of face-to-face interactions as described in the left

[11] `https://developers.google.com/drive/realtime`
[12] `https://www.dropbox.com`
[13] `https://www.firebase.com`
[14] Representational State Transfer [Fie00]
[15] `http://nosql-database.org`

side of the CSCW matrix. It tunnels the repository connection, thus decreases the communication delay and increases the communication reliability, especially in mobile situations or when the repository is not available. The extension is easy to integrate into ECP applications and enables model-based real-time synchronization on arbitrary EMF models. We additionally created an Objective-C version of this component which can be used in the development of MacOS and iOS applications and which is compatible to the EMFStore and to ECP applications developed within Eclipse.

Collaborators can use real-time collaboration in local networks via Bonjour or Bluetooth and over the Internet via Websockets to concurrently work on the same object graphs without the need to forbid concurrent modifications of the same objects on a process-level. Our extension supports the initial synchronization and checkout of model instances if users have different versions, directly over peer-to-peer without the need of a connection to the remote repository. It focuses on conflict avoidance and implements privileged access through a locking mechanism on object level.

We implemented an initial version of our approach and want to evaluate it in the upcoming months. We particularly like to investigate whether our approach prevents situations where model instances are not up-to-date and whether the amount conflicts and their severity are decreased. We then adapt our framework extension according to the user feedback and make it open source in 2014.

While the EMFStore platform is currently mainly used in development processes to maintain the most current version of model instances, we can also imagine the usage of this platform for arbitrary applications. Even a simple chat or task application can use our approach to allow users to collaborate in real-time. Complex features like branching and merging do not need to be exposed to the user. With our framework extension developers developers are able to integrate real-time communication easily into any application.

5. REFERENCES

[Bus99] Frank Buschmann. *Pattern oriented software architecture: a system of patters*. Ashish Raut, 1999.

[Cha09] Scott Chacon. *Pro git*. Apress, 2009. `http://git-scm.com/book`.

[Fie00] Roy Fielding. Representational state transfer. *Architectural Styles and the Design of Network-based Software Architecture*, pages 76–85, 2000.

[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 Conference on CSCW*, pages 207–217. ACM, 1994.

[HK13] Jonas Helming and Max Koegel. EMFStore, 2013. `http://eclipse.org/emfstore`.

[Joh88] Robert Johansen. *GroupWare: Computer Support for Business Teams*. The Free Press, New York, NY, USA, 1988.

[KH10] Maximilian Koegel and Jonas Helming. EMFStore: A Model Repository for EMF Models. In *Proceedings of the 32nd International Conference on Software Engineering*, ICSE, pages 307–308. ACM, 2010.

[KHS09] Maximilian Koegel, Jonas Helming, and Stephan Seyboth. Operation-based conflict detection and resolution. In *Proceedings of the ICSE Workshop on Comparison and Versioning of Software Models*, pages 43–48. IEEE, 2009.

[KNHH10] Maximilian Koegel, Helmut Naughton, Jonas Helming, and Markus Herrmannsdoerfer. Collaborative model merging. In *Proceedings of SPLASH*, pages 27–34. ACM, 2010.

[Kop11] Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer, 2011.

[MC08] Dennis McCarthy and Chris Crane. *Comet and Reverse Ajax: The Next-Generation Ajax 2.0*. Apress, 2008.

[NMBT05] Tien N Nguyen, Ethan V Munson, John T Boyland, and Cheng Thao. An infrastructure for development of object-oriented, multi-level configuration management services. In *Proceedings of the 27th international conference on Software engineering*, pages 215–224. ACM, 2005.

[Swi08] Travis Swicegood. *Pragmatic version control using Git*. Pragmatic Bookshelf, 2008.

[Tor13] Linus Torvalds. git, 2013. `http://git-scm.com`.