Enforcing Entailment Constraints in Offline Editing Scenarios for Real-time Collaborative Web Documents

Patrick Gaubatz[1], Waldemar Hummer[2], Uwe Zdun[1] and Mark Strembeck[3]

[1]Faculty of Computer Science
University of Vienna
{first.last}@univie.ac.at

[2]Distributed Systems Group
Vienna University of Technology
hummer@infosys.tuwien.ac.at

[3]Institute for Information Systems
WU Vienna
mark.strembeck@wu.ac.at

ABSTRACT

Real-time collaborative Web applications allow a multitude of users to concurrently work on a shared document. Especially in business contexts it is often necessary to be able to precisely define and restrict who is allowed to edit which data field of such a shared document. Existing solutions for enforcing such access control restrictions typically rely on a central service, the policy decision point. However, for use cases with unreliable or limited connectivity, such as mobile devices, a permanent connection to this centralized policy decision point can not be guaranteed. To address this problem, we present a novel approach that includes methods for client-side enforcement of access control constraints for offline users, and merging of offline changes, that enables users to edit such access constrained shared documents offline. We propose a generic conflict detection and resolution approach that attempts to resolve merge conflicts that are inherent to access constrained documents automatically while prioritizing online users and maximizing the number of filled out data fields in a document. In addition, we discuss and evaluate our approach via a prototype implementation.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Web Applications; D.2.2 [**Software Engineering**]: Design Tools and Techniques

Keywords

Document Merge, Conflict Detection, Conflict Resolution, Authorization, Access Control Enforcement

1. INTRODUCTION

Real-time collaborative Web applications such as Google Docs or Etherpad aim to efficiently support the joint work of team members, allowing them to collaboratively work on the same Web document at the same time. While such text based applications are probably the most popular example of collaborative Web applications today, their use in typical business contexts is limited as many business applications usually deal with strict, standardized forms with precisely specified text fields (see, e.g., [11]). Another crucial aspect of business applications is access control. The general goal of our work is to address this combination of form-based collaborative Web applications with access control; in this paper, we specifically focus on offline editing of real-time collaborative Web forms which have access control constraints.

Role-based access control (RBAC) [17] is the de-facto standard for access control in software systems. In RBAC, roles are used to model different job positions and scopes of duty within a system. These roles are equipped with permissions to perform tasks. Human users (subjects) are assigned to roles according to their work profile. For example, in a real-time collaborative Web document for the e-health domain only subjects assigned to the role *physician* shall be allowed to prescribe medications by filling in the corresponding text field of a document. To enforce a four-eyes principle, a second *physician* shall check and sign these prescriptions. In this example the role *physician* is equipped with both permissions, i.e., prescribing and signing medications. To prevent a single subject from performing both tasks and undermining the four-eyes principle we have to constrain these two tasks. Entailment constraints (see, e.g., [6, 20, 22]) provide means for placing such restrictions on the subjects who can perform a task x given that a certain subject has performed another task y. Mutual exclusion and binding constraints are typical examples for entailment constraints. For instance, a dynamic mutual exclusion constraint can be used to realize the four-eyes principle. In particular, it defines that two subjects must not perform two mutually exclusive tasks of the same Web document.

The decision if a specific subject should be granted the permission to perform a task (e.g, filling in a field) is called authorization decision. In the context of Web-based environments, authorization decisions are typically delegated to a central service, the Policy Decision Point (PDP) (see, e.g., [14]). As a consequence, using a Web application that is subject to access control usually requires a user's device to have reliable network connectivity (to be able to access the PDP). However, the increasing importance of mobile computing and mobile devices in the context of business applications demands solutions that enable users to continue editing (entailment constrained) collaborative Web documents even if a reliable network connection can not be guaranteed. Such unreliable network connections frequently occur if the user

SAC'14 March 24-28, 2014, Gyeongju, Korea.

is on an airplane, in a train, in a basement, or in a rural area. To the best of our knowledge, existing literature does neither consider the enforcement of entailment constraints for offline editing of collaborative Web documents, nor does it provide a systematic approach for merging actions that have been performed offline on a shared collaborative Web document that is subject to entailment constraints.

In this paper, we therefore propose a novel approach to support access control enforcement for offline users. In our approach, the offline changes on a Web document are authorized and recorded at the client-side. After re-establishing a network connection the corresponding changes are merged with the online, collaborative Web document. To handle potential violations of entailment constraints that may result from merging changes that were performed offline, our approach includes automated conflict detection and resolution procedures. In addition, our approach allows for prioritizing online performed changes (over changes that were performed offline) and maximizes the number of filled out data fields in a Web document.

The main contributions of this paper are as follows:

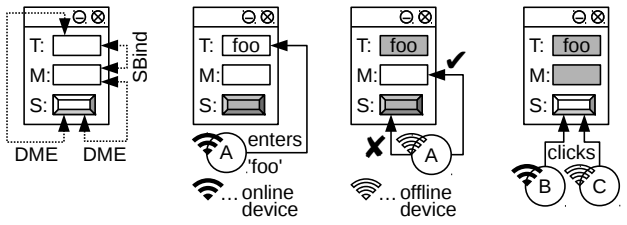
- We propose a system architecture that enables users of real-time collaborative Web applications to edit entailment constrained Web documents offline.
- We discuss how merge conflicts are inherent to editing such constrained Web documents offline and present a merge approach capable of detecting and resolving these conflicts automatically.
- We explain how the approach can be mapped to already existing Web browser technologies and APIs.
- We discuss performance and scalability measurements of the implemented prototype and evaluate the degree of automation of the proposed merge approaches.

The structure of this paper is as follows: Section 2 motivates our approach using an example scenario. Section 3 presents the conceptual details of our approach. We discuss our prototype implementation in Section 4 and evaluate the approach in Section 5. After a discussion of related work in Section 6, we conclude in Section 7.

2. MOTIVATING SCENARIO

For illustration, we consider a collaborative Web application for maintaining patient health records. Figure 1a shows a simplified form with two text fields (T and M) and a button (S). A *physician* is supposed to enter a therapy plan into text field T. Text field M is used to prescribe a specific medication. The hospital requires that only the *physician* who originally proposed a therapy plan is allowed to prescribe the medication. Entailment constraints (see, e.g., [6, 20, 22]) provide means for defining such restrictions. In this particular example, a subject-binding (SBind) constraint defines that the subject entering text field M must be the *same* that has filled in text field T. Moreover, to enforce a four-eyes principle, the hospital also requires prescriptions to be confirmed (button S) by a *different* physician. Hence, a dynamic mutual exclusion (DME) constraint is used to define that the subject filling in text field M must not be allowed to click button S.

We now consider subjects A, B and C who are concurrently editing this form. Figure 1b shows that Subject A,



(a) Constraints (b) Enforcement (c) Offline Enf. (d) Conflicts

Figure 1: Exemplary Collaborative Web Application

whose device is currently online, fills in field T. As a result, the system must disable text field S for Subject A (indicated by a gray background) in order to prevent violation of the DME constraint. Now, suppose that Subject A loses its network connection. Ideally, the Web application should allow Subject A to continue working offline, while continuing to enforce the entailment constraints (i.e., by allowing A to edit text field M and preventing A from editing text field S, see Figure 1c) even without a connection to the central PDP. In such a scenario, authorization decisions are (temporarily) made at the client side.

After Subject A has filled out T, both Subject B and C can potentially click S. Assume in Figure 1d that Subject C also (temporarily) loses network connectivity, while B confirms the prescribed medication by clicking S. Since C is offline and can not be informed of this action, C is also (locally) permitted to click the same button. Eventually, as Subject C re-establishes its network connection, the Web application is confronted with conflicting actions, because only a single confirmation can be stored in the document. The naive approach would be to enforce a first-come-first-serve principle by blocking C's action. However, this approach may be suboptimal if the global goal is to fill out as many form fields as possible. Moreover, it may result in violations of potentially defined entailment constraints. Thus, if we consider that C submits an entire set of actions performed offline, it may for instance be advantageous to revert B's changes and give precedence to C's.

In summary, we identify two main challenges:

- If a client loses connectivity, both the authorization decision making and the enforcement of entailment constraints have to be conducted at the client side instead of relying on the central server-side PDP.
- Offline editing of entailment constrained real-time collaborative Web documents may lead to conflicts that must be detected and resolved automatically.

3. OFFLINE EDITING APPROACH

Figure 2 provides an architectural overview of the components and interactions used to support offline editing for an entailment constrained collaborative Web document.

The left-hand column of the figure depicts the core components of a real-time collaborative Web application. A Collaboration Service allows users to concurrently work on the same collaborative Web document by constantly keeping the server-side Shared Model (i.e., the data model/content of a concrete Web document) in sync with all client-side Local Models (i.e., exact copies of the Shared Model).

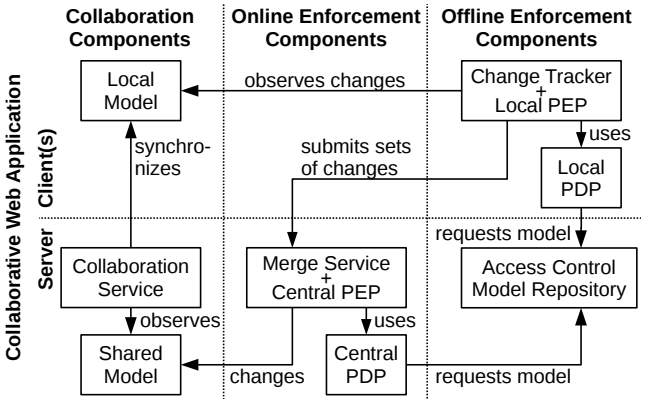


Figure 2: Architectural Overview

To prevent a user from unauthorized manipulation of protected parts of this Shared Model, our approach requires all model changes (i.e., changes of data fields within a Web document) to be routed through a Merge Service which acts as the central Policy Enforcement Point (Central PEP). This service enforces the authorization decision of the Central PDP. More specifically, the Merge Service only applies a change if the Central PDP confirms that the user actually has the permission to perform that change.

Supporting offline editing builds upon three basic ideas:

1. We persist the Local Model and duplicate the functionality of the Central PDP to the clients, allowing them to make authorization decisions locally (see Section 3.1).

2. The tracked changes of the Local Model on the clients are submitted to the central Merge Service as soon as the network connectivity is regained (see Section 3.2).

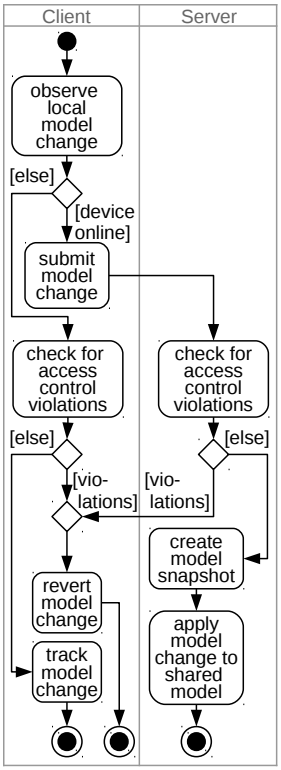
3. We strive to detect and automatically resolve conflicts that result from merging offline performed changes with the Shared Model (see Section 3.3).

3.1 Supporting Client-side Enforcement

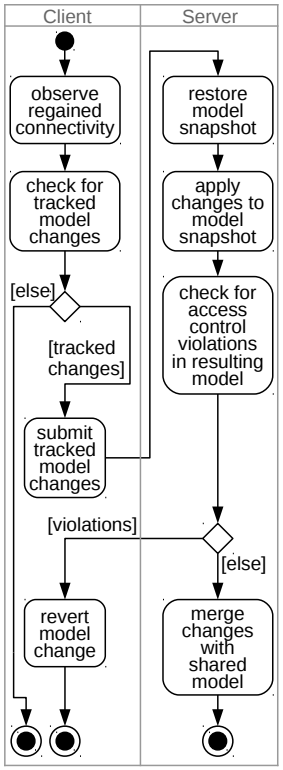
Figure 3a depicts the steps that are triggered whenever a user makes changes to their Local Model.

The Change Tracker component observes such changes. If the user is online, the change is directly submitted to the Merge Service, where it is checked for violations of entailment constraints and eventually applied to the Shared Model. In case of violations, the change is discarded and the corresponding data field in the client's Local Model is reverted to its previous state. Otherwise, if the user's device is offline, the Change Tracker requests an authorization decision from the Local PDP to determine if the change violates any constraints, in which case it gets reverted. Otherwise, the change is tracked and stored by the client until a merge with the Shared Model is possible.

The Central PDP and the Local PDP require the same entailment constraint models, which reside on a central Access Control Model Repository service. To guarantee that the user can continue working offline, the Local PDP requests and persists the current model state on client side. Ideally this is done as soon as a specific Web document is accessed for the first time.



(a) Steps that are triggered when a user changes a data field



(b) Steps that are triggered when a user regains network connectivity

Figure 3: Client-side Enforcement and Merging

3.2 Merging Offline Performed Changes

The Change Tracker tracks and stores model changes performed offline. Figure 3b shows the steps that are triggered as soon as a user regains network connectivity. First, the Change Tracker checks if (newly) tracked model changes exist. If no model changes were tracked the process ends. Otherwise, it submits the set of changes to the Merge Service.

Before merging the changes with the Shared Model, the Merge Service must first ensure that all offline changes were legitimate. This additional step is crucial to avoid illegitimate model changes (e.g., in case the Local PDP has been tampered with). Checking the legitimacy of changes in a document requires the Central PDP to be aware of *who* (e.g., Subject x using Role y) has performed which changes on a document. Therefore we propose a snapshot-and-replay approach where the Shared Model is versioned, such that a snapshot of the current state is created for every change. For each incoming merge request the Merge Service restores the corresponding snapshot of the Shared Model that a set of offline changes is based on. Finally, the Merge Service applies ("replays") these offline changes to the snapshot and checks the validity of the resulting model. Thereby, the integrity of the client-side Local PDP can be verified and only legitimate changes are merged with the Shared Model.

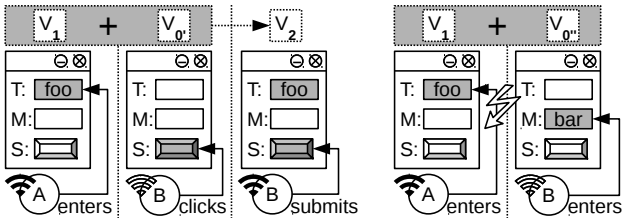
The actual merge procedure involves the Shared Model and a single client's set of offline changes. In essence, we apply each change to the corresponding data field of the Shared Model. To avoid data inconsistencies (if two clients simulta-

Conflict	Description
Duplicate Field	The same field is filled out twice.
Subject-binding (SBind) Violation	SBind constrained fields are filled out by different subjects.
Role-binding (RBind) Violation	RBind constrained fields are filled out using different roles.
Dynamic Mutual Exclusion (DME) Violation	DME constrained fields are filled out by the same subject.
Static Mutual Exclusion (SME) Violation	SME constrained fields are filled out by the same subject or using the same role.

Table 1: Potential Merge Conflicts

neously submit their changes), the Merge Service acquires a lock on the Shared Model. Simultaneously submitted merges are therefore serialized, i.e., processed sequentially.

For illustration, consider the exemplary merge situation depicted in Figure 4a (Figure 4b is discussed in Section 3.3). By filling out text field T, Subject A increases the Shared Model's version counter from V_0 (i.e., the empty document) to V_1. At the same time Subject B, which is offline and still working with V_0, clicks button S. Afterwards, Subject B submits this offline change and eventually (i.e., after restoring V_0, applying the change and validity checking the resulting model) text field T is merged with the current Shared Model's version V_1, resulting in the new merged version V_2.



(a) Non-conflicting Merge (b) Conflicting Merge

Figure 4: Exemplary Merge Situations

3.3 Detecting and Resolving Merge Conflicts

Offline editing for real-time collaborative Web documents may inevitably lead to merge conflicts. Let us reconsider the exemplary merge situation described above (see Figure 4). If we suppose that Subject B fills out text field M while working offline, we are confronted with a merge conflict situation that is depicted in Figure 4b. More specifically, Subject B's offline completed text field M can not be merged without violating the defined subject-binding constraint (see Figure 1a).

In general, a merge conflict happens either when the same field is filled in twice (i.e., online and offline) or when a merge results in a model that violates entailment constraints. We systematically analyzed the respective entailment constraint models (see, e.g., [6,20,22]) to compile a list of conflict types that may arise when merging data fields that are subject to such constraints. Table 1 depicts this set of conflict types.

Detecting merge conflicts is straightforward. However, resolving them (semi-)automatically is not. First of all, there is not a single "one-size-fits-all" resolution strategy (see, e.g., [18]). For instance, to resolve a *Duplicate Field* conflict we could choose between the following resolution strategies: (1) concatenate both (field) values, (2) try to merge both values into a single value, (3) move one value to an attachment and let a human person manually resolve the conflict, or (4) discard one value. However, Strategies 1 and 2 are not always sensible (e.g., if a subject-binding exists), and since Strategy 3 does not resolve conflicts instantaneously we will focus on Strategy 4 for the rest of this paper.

Name	Return Value
getConstraints	Set of constraints a *field* is subject to.
getConstrainedField	Returns the field that is bound to the same *constraint* as *field*.
getSubject	Subject that has filled out *field*.
getRole	Role that has filled out *field*.
violates	`true` if *field* and *anotherField* violate *constraint*.

Table 2: Helper Procedures for Algorithm 2 and 3

The main challenge of this strategy is to decide which values to discard. For instance, in Figure 4b we could discard Subject B's value for field M and leave the Shared Model unchanged. Alternatively, we could revert Subject A's field T and merge Subject B's field M instead. However, by reverting and ultimately deleting Subject A's field partially sacrifice the main idea of the real-time collaboration approach: keeping a document in sync for *online* users that are participating in an *online* collaborative session. Therefore, we allow that changes performed by online users can be prioritized over ones that have been performed by offline users.

Algorithm 1 Basic Merge Algorithm

```
1:  procedure BasicMerge(model, offChanges, offWeight)
2:      fieldsToDelete ← FieldsToDelete(model, offChanges)
3:      if |fieldsToDelete| ≤ |offChanges| × offWeight then
4:          for each fieldToDelete in fieldsToDelete do
5:              model ← model \ fieldToDelete
6:          end for
7:          for each fieldToMerge in offChanges do
8:              model ← model ∪ fieldToMerge
9:          end for
10:     end if
11: end procedure
```

To devise a generic merging approach based on the assumption that prioritization of online changes is crucial in the context of real-time collaboration, we introduce a decision criterion that determines whether a set of online changes should be deleted in order to be able to merge another set of offline changes. This decision is reflected in Algorithm 1. Note, that all required helper procedures are listed in Table 2. In our approach, a set of offline changes ($offChanges$) gets merged only if the set of fields that needs to be deleted ($fieldsToBeDeleted$) is smaller than the former. This approach ensures that a merge—if performed—*never decreases* the number of completed data fields in the Shared Model. To support prioritization of online completed fields we introduce a weighting factor ($offWeight$) that is used to discriminate the set of offline changes (as seen in line 3 of Algorithm 1). For instance, a factor of 0.5 means that removing one online completed field is equally bad as discarding two offline completed fields. Applying this to Figure 4b (i.e., $fieldsToDelete = \{\mathsf{T}\}$ and $offChanges = \{\mathsf{M}\}$) we get $1 \leq 1 \times 0.5$, decide *against merging* and *discard* the given set of offline changes.

Algorithm 1 relies on Algorithm 2 to determine the set of fields that would have to be deleted in order to merge

Algorithm 2 Fields-to-Delete Algorithm

```
1: procedure FieldsToDelete(model, offChanges)
2:     fieldsToDelete ← ∅
3:     for each field in offChanges do
4:         if field ∈ model then
5:             fieldsToDelete ← fieldsToDelete ∪ field
6:         end if
7:         for each constraint in getConstraints(field) do
8:             otherField ← getConstrainedField(constraint, field)
9:             if otherField ∈ model
                 and violates(field, otherField, constraint) then
10:                fieldsToDelete ← fieldsToDelete ∪ otherField
11:            end if
12:        end for
13:    end for
14:    return fieldsToDelete
15: end procedure
```

the set of $offChanges$ with the Shared Model. To account for *Duplicate Field* conflicts (see Table 1), Algorithm 2 first adds all fields (contained in $offChanges$) that have already been completed in the Shared Model to the $fieldsToDelete$ list. The remainder of the algorithm (i.e., lines 7–13) deals with conflicts that may occur due to violations of entailment constraints (see Table 1). For instance, regarding Figure 4b this means that merging Subject B's field M, requires deleting Subject A's field T. Otherwise, the subject-binding constraint that is associated with fields T and M (see Figure 1a) would be violated. In general, Algorithm 2 adds all fields to $fieldsToDelete$ that must necessarily be deleted from the Shared Model in order to allow $offChanges$ to be merged without violating any defined entailment constraints.

Algorithm 3 Two-step Merge Algorithm

```
1: procedure TwoStepMerge(model, offChanges, offWeight)
2:     constrained ← ∅
3:     for each field in offChanges do
4:         if getConstraints(field) ≠ ∅ then
5:             constrained ← constrained ∪ field
6:         end if
7:     end for
8:     unconstrained ← offChanges \ constrained
9:     BasicMerge(model, unconstrained, offWeight)
10:    BasicMerge(model, constrained, offWeight)
11: end procedure
```

Our merge algorithm (i.e., Algorithm 1) realizes an "all or nothing" approach, i.e., the set of offline changes is either completely merged or discarded. For instance, the algorithm might discard a complete set of offline performed changes just because the set contains a single constrained data field that required lots of the Shared Model's data fields to be reverted. Consequently, a key to improving this basic merge approach is to split the set of offline changes up into smaller subsets and merge these subsets with the Shared Model one after another. To this end, we propose a two-step approach (see Algorithm 3) that first tries to merge the distinct set of unconstrained data fields exclusively. Afterwards, the remaining set of constrained data fields is merged. As we show in Section 5 the two-step merge approach exhibits a significantly higher chance that a set of offline changes can be merged than basic merge. However, for large numbers of fields the basic merge approach yields a better performance.

4. IMPLEMENTATION

This section discusses a prototype implementation that has been developed to prove the feasibility of our approach. It is founded on CoCoForm, a Web application framework for real-time collaboration, supporting access control [10,11].

Our approach is completely decoupled from the collaborative aspects of the application. Thus, it is complementary to currently available frameworks for the development of real-time collaborative Web applications, such as ShareJS[1] or the Open Cooperative Web Framework[2] (OpenCoweb). In fact, our prototype extends OpenCoweb, which consists of both, a Collaboration Service (as in Figure 2) and a JavaScript API that allows us to keep the Shared Model in sync with all client-side Local Models. Both, the Change Tracker and the Local PDP issue simple `XMLHttpRequests` to submit local model changes to the central Merge Service and to request entailment constraint models from the Access Control Model Repository. In case of failed requests (i.e., `timeout` or `error` events are emitted) the Change Tracker uses the Local PDP to determine if a change should be tracked or not. We implemented the Local PDP (i.e., in JavaScript) in such a way that it exhibits a runtime behavior that is identical to the Central PDP's (which has originally been implemented in Java). The Change Tracker and the Local PDP leverage the Web Storage API[3]. More specifically, the `window.localStorage` JavaScript object provided by this API is used to permanently persist tracked model changes, the required entailment constraint model as well as the Local Model in the user's browser. The Change Tracker uses the `window.navigator.onLine` attribute as well as the emitted `online` event to react to regained network connectivity. The Application Cache[4] feature of HTML5 permanently caches all required assets (e.g., JavaScript or CSS files) upon accessing the Web application for the first time. This allows it to be accessed and executed in offline mode.

All server-side components are implemented in Java. More specifically, the Merge Service and the Access Control Model Repository are implemented as plain HTTP Services using the JAX-RS API. Internally we work with Ecore[5] model instances that are marshalled into JSON for the client-side application code. We use a model-driven approach for defining forms and documents and securing them using entailment constraints. Note that the corresponding meta-models have been presented in our previous work [11].

5. EVALUATION

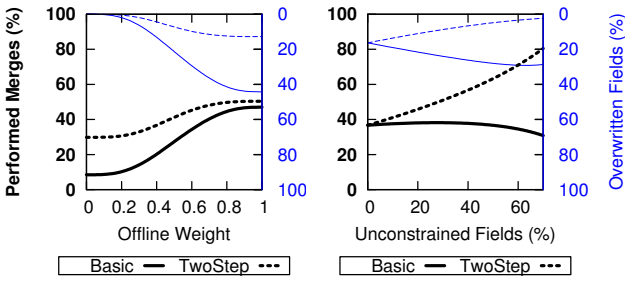
In this section we first evaluate the merge algorithms' possible degree of automation as well as the runtime performance and scalability of a prototype implementation. Then, we evaluate the performance and scalability of our prototype Merge Service. Finally, we contrast two implementation variants of our proposed snapshotting approach.

5.1 Evaluation of the Merge Algorithms

A crucial part of our approach both in terms of the possible degree of automation and the overall performance are the merge algorithms, which we evaluate in this section.

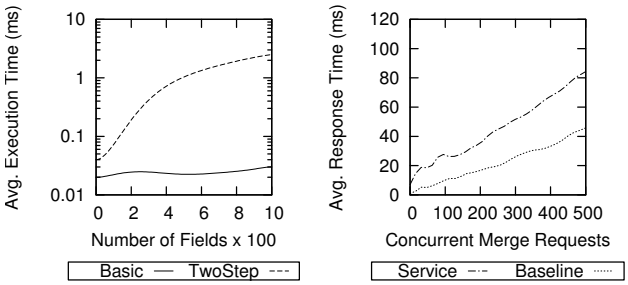
We evaluate the merge algorithms' possible degree of automation by analyzing and comparing the percentage of actually *performed merges* and of *overwritten fields*. The eval-

[1] `http://sharejs.org`
[2] `http://opencoweb.org`
[3] `http://www.w3.org/TR/webstorage`
[4] `http://www.w3.org/TR/html5/browsers.html#offline`
[5] `http://www.eclipse.org/modeling/emf`



(a) Varied Offline Weights (b) Constrained vs. Unconstrained Fields

Figure 5: Evaluation of Two Merge Algorithms



(a) Algorithms (b) Merge Service

Figure 6: Performance Evaluations

uations have been performed using an exemplary, typical document and entailment constraint model with two subjects, one role and four data fields. Three of the data fields are constrained by subject-binding constraints and the remaining field is unconstrained. For this model, we have calculated all permutations of possible merge combinations. For these combinations, first, we contrast the basic (i.e., Algorithm 1) and two-step merge algorithms (i.e., Algorithm 3), the actually performed merges, and overwritten fields for different values of the offline weight parameter. Second, we contrast the performed merges and overwritten fields for different portions of unconstrained fields to analyze the impact of changes in the document and entailment constraint model.

Our approach introduces the notion of an offline weight as a means of discriminating offline changes (see Section 3.3). Figure 5a depicts how the offline weight affects the automatability, i.e., the chance that a merge can be performed automatically. For instance, in our exemplary scenario and an offline weight of 0.4 nearly 16.5% of all possible merge combinations could be merged using the basic merge algorithm, while 33% could be merged using the two-step merge algorithm.

On the other hand, a growing offline weight value leads for both algorithms to more deleted and eventually overwritten fields of the Shared Model. Figure 5a provides evidence that – at least in our exemplary scenario – the two-step merge algorithm always performs better than the basic merge algorithm, both in terms of a higher number of performed merges and a lower number of overwritten fields.

While the exact progression of the graphs is specific to our exemplary model, similar tradeoff graphs can be calculated for other models. That is, on the one hand, a higher offline weight increases the automatability, and, on the other hand, it also increases the number of situations where online data fields have to be overwritten. Consequently, there is no universally optimal offline weight, but it has to be determined empirically for a specific document and entailment constraint model.

To illustrate and analyze the impact of changes in the document and entailment constraint model, we set the offline weight parameter to 0.5 and change the number of unconstrained fields in the document. In Figure 5b we can see the effects of changing the ratio of constrained vs. unconstrained fields in our exemplary scenario. Obviously, for a model that has no constrained fields at all, the results are identical for both algorithms. If half of all fields in our exemplary model are unconstrained, the two-step merge algorithm manages to merge nearly twice (i.e., 62%) as often and exhibits nearly a seven times lower chance of overwritten fields compared to the basic merge algorithm. The gap between the two algorithms gets even wider with a growing ratio of unconstrained fields. In summary, we observed that in both scenarios (i.e., with changing offline weights and especially with a growing number of unconstrained fields in a model) the two-step merge algorithm exhibits significantly better characteristics than the basic merge algorithm.

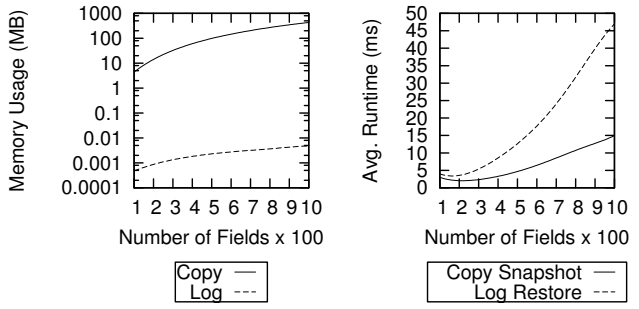
Algorithm 2 has a worst-case runtime complexity of $O(N \times M)$, i.e., in simplified form we can write $O(N^2)$. As a consequence, both merge algorithms (i.e., Algorithm 1 and 3) also have a runtime complexity of $O(N \times M)$.

Because of the quadratic complexity and because Algorithm 3 invokes Algorithm 1 two times, we further studied the performance impact of both approaches. The presented measurements are based on our prototype implementation (see Section 4) and have been conducted on a machine equipped with a 2.4 GHz dual core CPU, 8 GB RAM, running Ubuntu GNU/Linux 13.04. The performance is mainly dependent on the number of fields in a document. Figure 6a visualizes the average execution time of a merge operation for various field counts. Note that the y-axis is scaled logarithmically. Although the measurements exhibit a significantly higher average execution time for the two-step merge algorithm, the performance penalty should be negligible for "reasonable" field counts. We consider the execution times for both approaches to be acceptable, especially when keeping in mind that merges typically happen only occasionally.

5.2 Evaluation of the Merge Service's Performance and Scalability

In addition to the performance evaluations for the merge algorithms, we analyzed the performance of our prototype Merge Service. In particular, we analyzed how well our Merge Service handles merge requests that are issued simultaneously by a potentially large number of users.

Figure 6b compares the average response times of the Merge Service (using the two-step merge algorithm) and a "Baseline Service" (i.e., no computation at all), for a given number of simultaneous requests. For instance, in the case of 250 requests, the average response time for all clients is roughly 40ms and 20ms for the Baseline Service. This means, that in this case it takes roughly 20ms to perform the actual merge operation, while the remaining 20ms account for the underlying communication and Web Service



(a) Memory Consumption (b) Execution Time

Figure 7: Evaluation of Snapshotting Approaches

stack. Our results indicate linear scalability and even in the case of 500 users simultaneously submitting their offline changes, the average response time remains below a tenth of a second. As the Baseline Service represents the theoretical minimum that is possible for the given Web Service framework, we consider the performance overhead of our Merge Service to be acceptable in most scenarios.

Note that we used the same machine and environment as for the measurements of the merge algorithms. Additionally, both the services and the testing tool, i.e., Apache's `ab` tool[6] ran on the same machine. Hence, the measurements are free from network-induced effects such as latency or jitter.

5.3 Evaluation of Snapshotting Approaches

In Section 3.2 we motivated the need for a versionized Shared Model and proposed a snapshot-and-replay approach. A document with n data fields potentially requires n snapshots (i.e., if the fields are filled out one after the other). Hence, we measured the consumption of computing resources with a growing n. More specifically, we contrast two implementation variants. First, the *Copy* approach stores a clone of the complete document for each snapshot. Second, the *Log* approach merely stores a log entry (i.e., who changed which fields). Figure 7a illustrates the amount of memory that is required to hold n snapshots of a document of n data fields in memory for both approaches using our prototype implementation. Obviously, the *Copy* approach requires approximately four orders of magnitude more memory than the *Log* approach.

The corresponding (average) time that is needed to create a single snapshot (i.e., to clone a document) using the *Copy* approach is depicted in Figure 7b. As creating a snapshot using the *Log* approach has a runtime complexity of $O(1)$ we can neglect it. On the other hand, using the *Log* approach it takes significantly longer to restore a snapshot (i.e., the Shared Model has to be cloned and all changes that have been performed after the snapshot version to-be-restored have to be reverted in the cloned model) than for the *Copy* approach (which is $O(1)$). In summary, we conclude that the *Log* approach is considerably more resource efficient than the *Copy* approach.

6. RELATED WORK

In this section we discuss existing work in the related areas of Web collaboration platforms, access control enforcement, and (Web) document consistency.

Web Collaboration Platforms. Systematic development of Web collaboration platforms has received considerable attention. Heinrich et al. [13] propose a collaboration infrastructure aimed at transforming single-user Web applications into collaborative multi-user applications by synchronizing DOM trees. Farwick et al. [8] discuss an architecture for Web-based collaborative meta-modeling. Their framework allows multiple users to work on graphical meta-models collaboratively. Modifications of the (meta-)models are secured by basic access control measures, but in contrast to our work, they do not explicitly address dynamic updates resulting from merging offline actions under RBAC entailment constraints. Our work builds on frameworks and libraries that facilitate the development of collaborative Web applications. For instance, the Open Cooperative Web Framework (see Section 4) consists of a set of JavaScript libraries and a generic Java servlet. MobWrite[7] is another approach for enabling real-time collaboration. However, it is restricted to synchronizing HTML forms, and the re-usability and applicability is thus somewhat limited.

Security and Access Control Enforcement. Throughout the last years, a plethora of approaches have been presented for integrating security and access control in Web applications. Joshi et al. [15] provide an early study on generic security models for Web-based applications. In [4], Belchior et al. model RBAC policies using RDF triples and N3Logic rules. Tackling security on the client side, Guarnieri et al. [12] propose the GATEKEEPER framework for authoring and enforcing policies in JavaScript code. This complements our approach of having a local PDP component on the client devices. Ahn et al. [1] present an approach for injecting RBAC into an already existing Web-based workflow system. They propose a special reverse proxy for enforcing RBAC rules transparently to the actual Web application. Sohr et al. [19] and Hummer et al. [14] propose a similar approach in the context of Web services, using interceptors that are able to prevent the actual invocation in case of a policy violation. Several works on distributed security enforcement, particularly in mobile networks, complement and have influenced our approach. For instance, Alicherry et al. [2] support offline nodes by early distribution of policy tokens among network nodes. Similarly, Gasmi et al. [9] partition networks into trusted virtual domains which can operate and manage access rights independently. Finally, while we focus on access control, various other issues and threats also need to be taken into account, as studied by Almorsy et al. [3]. Their approach utilizes Object Constraint Language to define vulnerability signatures for threats like SQL injections or cross site scripting. Based on the signatures, different mitigation actions are proposed.

Document Consistency. The seminal work of Sun et al. [21] proposes the transparent adaptation (TA) approach to develop collaborative multi-user applications. The cornerstone of TA is operational transformation (OT) [7]. Given two concurrent operations o_1 and o_2 resulting in states s_1 and s_2, the core idea of OT is to transform the parameters of the operations to execute them on the current state while maintaining document consistency. Our approach is orthog-

[6] `http://httpd.apache.org/docs/2.4/programs/ab.html`

[7] `https://code.google.com/p/google-mobwrite`

onal to OT: the RBAC policies and entailment constraints provide an application workflow with well defined responsibilities, and we maintain document consistency by allowing only sequences of operations that comply with this workflow. The problem of consistency with concurrent modifications has also been a core issue in database research. The general approach of allowing concurrent work, followed by merging and resolving conflicts is denoted *optimistic concurrency control* [5] in databases. In this sense, the sequence of offline actions in our approach relates to a transaction in databases, and during merging the action sequences "compete" with the online changes that have happened in the meantime. A consistent merging strategy that could be applied to this problem is discussed and formally verified by Lin and Mendelzon in [16].

7. CONCLUSION AND FUTURE WORK

In this paper we have shown that offline editing for entailment constrained, real-time collaborative Web documents can effectively be realized using a combination of client-side access control enforcement and a document merging approach. We highlighted that merging such documents is inherently prone to conflicts and motivated the need for a merge approach that is capable of detecting and resolving conflicts automatically. We provided evidence that many possible conflicts can be resolved automatically and that both the merge algorithms and the prototype Merge Service work with acceptable runtime performance and scalability even for lots of simultaneous merge requests and documents with lots of data fields. In addition, we demonstrated that modern Web browsers as well as HTML5 and some of its accompanying APIs provide a platform that can be used to implement our novel approach.

Future work will address limitations inherited from HTML5 and Web browser implementations, such as the limited client-side storage capacity which be problematic if we have to deal with huge document and entailment constraint models (i.e., tens of thousands of model elements). Although we have shown that our basic merge approach works quite well, the characteristics of the two-step merge approach leads us to believe that there is still room left for improvements in that area. Another interesting topic would be to devise an approach for estimating (i.e., instead of determining it empirically) the optimal offline weight for a given document and the corresponding entailment constraint model. Furthermore, we will apply our approach to other types of collaborative processes. In particular with regard to dynamic processes (e.g., free text editing or modeling) we will have to deal with completely dynamic document and access control and constraint models (i.e., models that change at runtime).

8. REFERENCES

[1] G.-J. Ahn, R. Sandhu, M. Kang, and J. Park. Injecting RBAC to secure a Web-based workflow system. In *5th ACM workshop on RBAC*, pages 1–10, 2000.

[2] M. Alicherry, A. Keromytis, and A. Stavrou. Deny-by-default distributed security policy enforcement in mobile ad hoc networks. In *5th SecureComm*, 2009.

[3] M. Almorsy, J. Grundy, and A. S. Ibrahim. VAM-aaS: Online Cloud Services Security Vulnerability Analysis and Mitigation-as-a-Service. In *13th WISE*, 2012.

[4] M. Belchior, D. Schwabe, and F. Silva Parreiras. Role-based access control for model-driven web applications. In *12th ICWE*, pages 106–120, 2012.

[5] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[6] E. Bertino, E. Ferraria, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *TISSEC*, 2(1), 1999.

[7] C. Ellis and S. Gibbs. Concurrency control in groupware systems. *SIGMOD Record*, 18(2):399–407, 1989.

[8] M. Farwick, B. Agreiter, J. White, et al. A web-based collaborative metamodeling environment with secure remote model access. In *10th ICWE*, 2010.

[9] Y. Gasmi, A.-R. Sadeghi, et al. Flexible and secure enterprise rights management based on trusted virtual domains. In *3rd ACM STC workshop*, 2008.

[10] P. Gaubatz, W. Hummer, U. Zdun, and M. Strembeck. Supporting customized views for enforcing access control constraints in real-time collaborative web applications. In *13th ICWE*, 2013.

[11] P. Gaubatz and U. Zdun. Supporting entailment constraints in the context of collaborative web applications. In *28th Symposium On Applied Computing*, 2013.

[12] S. Guarnieri and B. Livshits. GATEKEEPER: mostly static enforcement of security and reliability policies for javascript code. In *USENIX Security'09*, 2009.

[13] M. Heinrich, F. Lehmann, T. Springer, and M. Gaedke. Exploiting single-user web applications for shared editing: a generic transformation approach. In *21st Int. Conf. on WWW*, pages 1057–1066, 2012.

[14] W. Hummer, P. Gaubatz, M. Strembeck, U. Zdun, and S. Dustdar. An integrated approach for identity and access management in a SOA context. In *16th ACM SACMAT Symposium*, pages 21–30, 2011.

[15] J. Joshi, W. Aref, A. Ghafoor, and E. Spafford. Security models for web-based applications. *Communications of the ACM*, 44(2):38–44, 2001.

[16] J. Lin and A. O. Mendelzon. Merging databases under constraints. *Int. J. of Coop. Inf. Systems*, 07(01), 1998.

[17] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *Computer*, 1996.

[18] S. Schefer, M. Strembeck, J. Mendling, and A. Baumgrass. Detecting and resolving conflicts of mutual-exclusion and binding constraints in a business process context. In *19th CoopIS*, 2011.

[19] K. Sohr, T. Mustafa, X. Bao, and G.-J. Ahn. Enforcing Role-Based Access Control Policies in Web Services with UML and OCL. In *24th ACSAC*, 2008.

[20] M. Strembeck and J. Mendling. Generic algorithms for consistency checking of mutual-exclusion and binding constraints in a business process context. In *18th CoopIS*, pages 204–221, 2010.

[21] C. Sun, S. Xia, et al. Transparent adaptation of single-user applications for multi-user real-time collaboration. *ACM TOCHI*, 13(4):531–582, 2006.

[22] J. Wainer, P. Barthelmes, and A. Kumar. W-RBAC - A workflow security model incorporating controlled overriding of constraints. *Coop. Inf. Syst.*, 12(4), 2003.