Universitat Politècnica de Catalunya

OSS Community involvement to provide a Thunderbird extension for different languages spell-checking

Final project

Author: Jordi Beltran Ràfols

Supervisor: Claudia Patricia Ayala Martinez
Department of Service and Information System Engineering.
ESSI

Degree in Informatics Engineering (2003)
Barcelona School of Informatics (FIB)

June 21th, 2016

Acknowledgements

I would like to thank the Mozilla Foundation for all their work on promoting openness, innovation and participation on the Internet. Maintaining Thunderbird and all other open source alternatives.

Also the Mozilla Thunderbird community, that keeps Thunderbird strong as is. And especially to the AMO reviewers, who dedicate their time to it, and therefore to us, the users of Thunderbird.

Also to the users that are using this extension but specially to the ones that did an extra effort and contacted me, reporting bugs, writing reviews or suggesting new features. This project would not be the same at all without their feedback.

The Open source community as a whole. It allows us to go further and dispose of technology and libraries that would not be at reach to a lot and would slow us down as humanity.

And finally to Claudia Ayala for all her support, patience and orientation.

Table of Contents

1. Introduction

The email is the main professional written communication tool between companies.

When this project was started in 2009, desktop clients were the main client types, but after the arrival of web and mobile clients, the last one has become the leader.

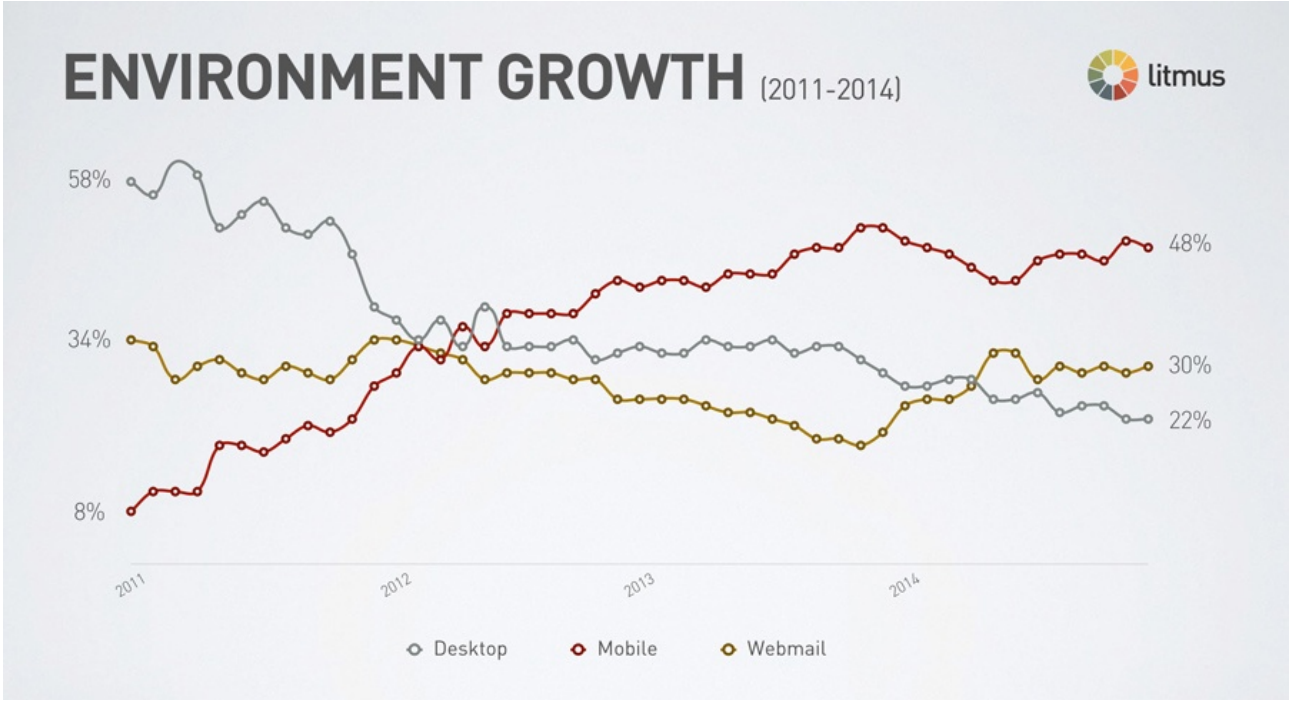


Illustration 1: Email clients evolution (source: litmus.com [1])

But stats collected by this extension also show that Thunderbird is used mainly on working hours, so we could say that Thunderbird is used mainly on professional environments.

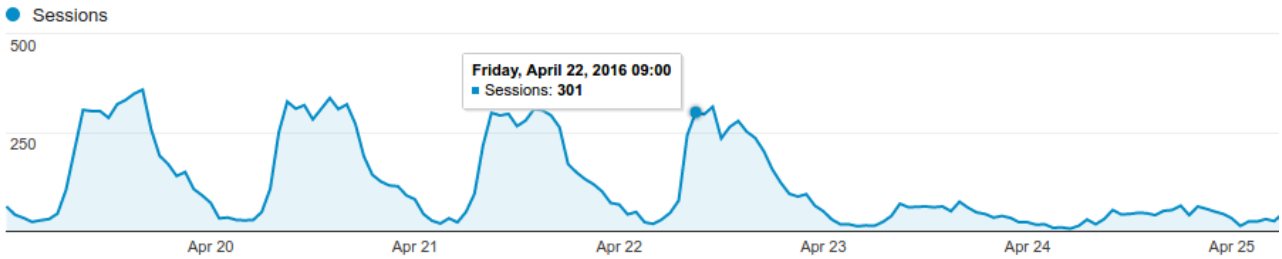


Illustration 2: Extension usage stats showing working hours trend

Even though desktop clients have decreased, they are still an important part.

1.1. Context

In this context of professional email clients, we face the fact that companies and economy as a whole, are more multilingual and international every day.

Users face situations where they have to write emails to different people in several languages, and have to think, when composing, which language should they use for those recipients. It's a little thinking that is made almost automatically, that usually finishes with the user switching the spell checker to the right language.

This little action takes time every day and this is where this work wants to help.

1.2. Motivation

In the context of the problem described above, I realized that, given that I was keen on open source and had been using it for a long time, it was a good chance to contribute to it. The need I detected could be a good addition and users may find it useful too.

I was specially interested in Mozilla organization and their fights for a free and open standards web. Also in learning how these contributions worked and which processes I was supposed to follow to achieve it.

And as I was using Thunderbird as my regular email client, I wanted to give back part of the great service it had been giving to me.

I knew I would have to learn the internals of Thunderbird and how extensions were supposed to work, but at first glance the source of Thunderbird seemed well documented and structured. It was a challenge but I could learn interesting things about it.

1.3. Objectives

Given that I had no experience in this community, my objective was to know how the community works and contribute in making the application switch the spell checker dictionary automatically.

I had to adapt to the community, to the way they expect contributions and the

flows defined to do so.

I also wanted to gain users, to get feedback and verify that it gave real value to them.

1.4. Related work

Back in 2009 there were no Thunderbird extensions that did the exact same thing I wanted, but there were some that tried to help on the same context.

I'll describe what they cover and what they don't and explain what I wanted.

1.4.1. Quick locale switcher

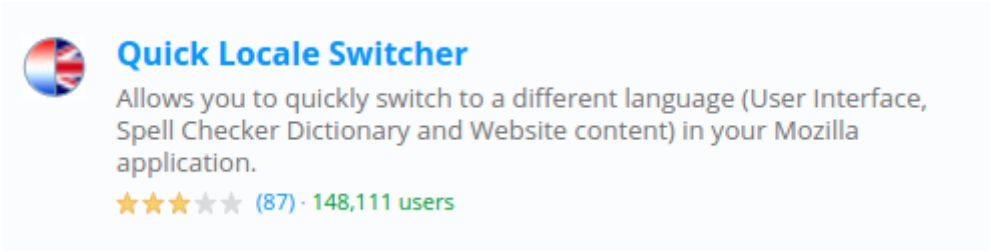


Illustration 3: Quick Locale Switcher add-on

https://addons.mozilla.org/en-US/thunderbird/add-on/quick-locale-switcher/?src=cb-dl-users

This add-on detects the language of the context (The text that surrounds the email body) and change the interface and spell-checker language to it.

In Thunderbird, that text is the existing text of the email. And in Firefox (the web browser) it takes into account the text of the current web page.

This add-on is useful on a website environment (in Firefox, i.e.) but on Thunderbird we sometimes we do not have context to infer the language from.

This add-on is also very used for developers to change the interface locale easily.

Update: between 2009 and 2015, this extension has added support to changing the spell-checker based on the email of the recipients, but it was not this way in 2009. Even now, it does not support the association of a locale to a

“recipient combination”. Only a single email address to a locale.

1.4.2. Dictionary switcher

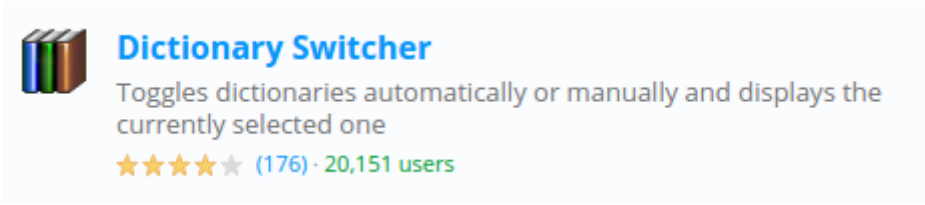


Illustration 4: Dictionary Switcher add-on

https://addons.mozilla.org/en-US/thunderbird/add-on/dictionary-switcher/?src=cb-dl-users

This add-on does the same as the Quick Locale Switcher but in this case it does not change the application interface. It only affects the dictionary. It also detects the language of the dictionary based on the language you are using on the current email. This means that you need to start to write the email before the plug-in can detect the right language.

This plug-in could cover most of the use cases our work would do, because it saves you to manually choose the language every time you write an email with a different language.

But in this case, it is not covering the case when the user chose the language wrong by mistake. When a user is writing an email with a wrong language, meaning wrong that some of the recipients do not understand it, it does not notify the user in any way. This is a reason why deciding the language on the recipients is better than on the email body.

1.4.3. Summary

This table describes which use case they cover and which they don't. The main difference here is that these plug-ins do not associate a language to a person.

Set the right language when...*	Quick locale switcher	Dictionary switcher
Replying to an email	Yes	Yes
Writing a new email	No	No
Write a new email (when you have written enough text)	Yes	Yes

Point the user that he may have chosen the wrong language for that recipient.	No	No

* And the expected language is different from the current.

Why there aren't more add-ons about this?

If you look at the problem they resolve, it seems clear that you could live without it, so it's something most users do not ask for.

If users get used to it, they will eventually ask for it on all email clients. But by now, these tools are installed in a little fraction of all clients. The add-on of this type that has more users is Quick Locale Switcher with 240k. And this represents only a 2.4% of all the users.

1.5. Structure of this document

This document is structured in several chapters and each chapter describes a different aspect or step of the project.

- Chapter 2, “Project planning” describes which methodology and planning was designed and its associated costs.
- Chapter 3, “Analysis and specification”, describes the problem and specify the requirements that will be covered to solve them.
- Chapter 4, “Design”, explains how we design a solution that fits the requirements defined in chapter 3.
- Chapter 5, “Implementation”, explains the implementation based on the solution designed.
- Chapter 6, “Verification”, describes how the solution applied performs and how the community reacted to this.
- Chapter 7, “Conclusions”, describes which conclusions can we deduce form the project, and which things did work and which did not.
- Chapter 8, “Bibliography”, lists all the references used in this document.

2. Project planning

This chapter describes how the project was planned, which methodology has been used and how it was performed compared to the planned schedule.

2.1. Methodology

We needed a methodology that fitted to the project needs and flows. In our case I knew that I needed to comply with the rules Mozilla has defined to contribute with an extension to their community. I knew that I needed to pass a validation (AMO certification) and they could ask me to change things.

After that, the validation would also be the user feedback. Once you release a version, users get updated, and if there is something wrong, they probably will contact you. Also, user validation would give us valuable feedback to know if we were in good direction.

We also wanted to choose the features based on data as much as possible, and not on intuition or guessing. As there was no final client expecting the product, we could decide what feature to release and when, there was no deadline or final date.

For those reasons we knew that an agile methodology would fit this project.

Our iterations weren't time scoped because my dedication to the project depended on factors outside this project. The iterations only were feature scoped.

For those reasons I chose Extreme Programming [2].

Each chapter explains the content, sliced in iterations when necessary, to describe the changes along them.

2.1.1. Requirement analysis

Given the current state of the problem, we have to understand it and decide the goals that we want to achieve to improve that situation. These goals will produce features to design and implement.

2.1.2. Software design

In this phase we design how we will give support to the requested features that will be implemented later. Describing which components will interact with and how.

2.1.3. Implementation

Once we have the design, we can implement the features and produce a working prototype.

2.1.4. Testing and validation

Once we have the prototype or the artifact with the new features, we check that everything is fine, nothing is broken, and we release it to the community. This step will bring real feedback of the decisions taken in each iteration.

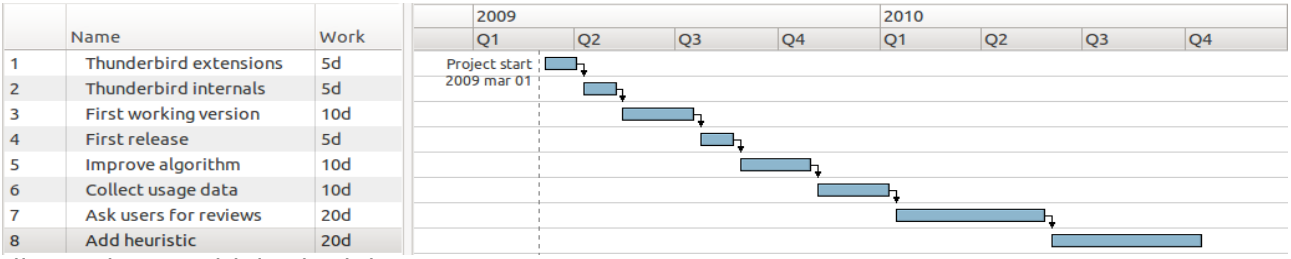
2.2. Schedule

As the methodology was agile, there was an initial road map that was going to be confirmed or modified on each iteration.

The initial road map was:

1. Iteration 1: Create a first version (non public) that does basic features.
2. Iteration 2: Release a first version and let users install it.
3. Iteration 3: Implement more sophisticated logic of remembering.
4. Iteration 4: Collect usage data to have global feedback.
5. Iteration 5: Ask users for reviews.
6. Iteration 6: Implement heuristics to match when we have no previous data.

This road map had no specific deadline. We'd give users time to give feedback after each upgrade, and give time to me to decide next iteration thoughtfully.



	Name	Work
1	Thunderbird extensions	5d
2	Thunderbird internals	5d
3	First working version	10d
4	First release	5d
5	Improve algorithm	10d
6	Collect usage data	10d
7	Ask users for reviews	20d
8	Add heuristic	20d

Illustration 5: Initial schedule

2.3. Final schedule

The resulting schedule has diverged mainly for these two reasons:

1. The estimated time for each task was optimistic sometimes and pessimistic others.

2. We did not take into account the time we spent fixing users bugs. This took a lot of effort because sometimes we did not have enough information from the user to reproduce the bug.

Iterations done:

1. First version for personal use

2. v1.0.x: First public release

3. v1.1.x: Limit stored data size and add more locales

4. v1.2.x: Map groups of emails to language and add preferences window.

5. v1.3.x: Google analytics, heuristic algorithm, and concurrency data locks.

6. v1.4.x: no-restart support, conversations add-on support, ask users for reviews.

7. v1.5.0: Let users say “no” to data collection easier.

8. v1.5.5: Bug fixing release.

Name
First private version
v1.0.x: First public release
v1.1.x: Limit stored data size and more locales
v1.2.x: Map groups of emails to language and preference window.
v1.3.x: Google analytics, heuristic algorithm, and concurrency data locks.
v1.4.x: no-restart support, conversations addon support, make users promote it.
v1.5.0: Let users say "no" to data collection.
v1.5.5: Fixing release
2009
Q4
Q1
Q2
Q3
Q4
2010
Q1
Q2
Q3
Q4
2011
Q1
Q2
Q3
Q4
2012
Q1
Q2
Q3
Q4
2013
Q1
Q2
Q3
Q4

2.4. Cost study

Considering the costs this project would have in the business environment, we describe two concepts, the tools required for the project, and the working hours.

2.4.1. Tools

The tools used on this project are all the physical or logic equipment that are not a human resource.

This includes a laptop to work, the software used to accomplish that, the power supply, and the Internet connection.

All these resources will be described and described how we calculate its cost.

Laptop

A laptop has an average cost of 700€ and as it's expected live is 4 years, and the project about this time, we could say that all the cost was fully amortized.

Tool	**Cost**
Laptop	700,00 €
Ubuntu Operating System	0,00 €
Emacs editor	0,00 €
Mozilla Thunderbird	0,00 €
Rhino JavaScript engine	0,00 €
Add-on review	0,00 €
	0,00 €

2.4.2. Human resources

During the project there have been different roles involved. The main is the developer, but it would be worthless without the rest.

We'll describe in each phase how many hours were spent and from which role. This data do not include the time some users spent testing our tool, but there have been several beta testers that have dedicated their personal time to give us very valuable feedback.

Also, the Mozilla community, has a team named “The AMO Reviewer Team” that has reviewed all our releases for free. They review new and updated add-ons as part of the add-on review process, and decide if they are appropriate for publication or not. They do not charge for this but they are worth mentioning.

Phase	Work done	Man-hours
First version for personal use	Read tutorials, and examples of extensions, Thunderbird code base, coding styles and create first prototype.	110
v1.0.x: First public release	Fix bugs from first version, code refactor, localization, and improve integration with Thunderbird	100
v1.1.x: Limit stored data size and more locales	Limit extension data stored and added Spanish and Catalan.	90
v1.2.x: Map groups of emails to language and preference window.	Remember languages for groups and allow to discard mails with too much recipients.	40
v1.3.x: Google analytics, heuristic algorithm, and concurrency data locks.	Collect analytics data, implement heuristic and add concurrency locking.	120
v1.4.x: no-restart support, conversations add-on support, ask for users reviews.	Plug-in does not need to restart Thunderbird to work, works with conversations, and asks users for reviews.	40
v1.5.0: Let users say “no” to data collection.	Ask all users if they are OK to collect statistical data on Google Analytics.	20
v1.5.5: Fixing release	Receiving feedback from several users that needed to be attended.	80
Document	Time spent composing this document.	80
		680

And the dedication of each role has been the following:

Phase	Analyst	Programmer	Tester	Total hours
First version for personal use	30	60	20	110
v1.0.x: First public release	20	70	10	100
v1.1.x: Limit stored data size and more locales	20	65	5	90
v1.2.x: Map groups of emails to language and preference window.	10	25	5	40
v1.3.x: Google analytics, heuristic algorithm, and concurrency data locks.	20	90	10	120
v1.4.x: no-restart support, conversations add-on support, ask for user reviews.	5	30	5	40

v1.5.0: Let users say “no” to data collection.	4	14	2	20
v1.5.5: Fixing release	10	60	10	80
Document	20	30	30	80
	139	444	97	680

We have divided the working hours by the following roles:

- Analyst: Responsible for defining requirements and designing a solution to the problem. The cost of this role is 35€/hour
- Programmer: The implementer of the solution. The cost of this role is 24€/hour
- Tester: the responsible for checking that there has been no regression and the features work as expected. The cost of this role is 20€/hour.

The hour of each role includes all the expenses related to their work. This means electricity consumption, taxes, office costs (if existed) and Internet connection fee.

Concept	**Hours**	**Cost per hour**	**Cost**
Analyst	139	35	4865
Programmer	444	24	10656
Tester	97	20	1940
Tools costs			
Laptop	1	700	700
			18161

3. Analysis

3.1. Stakeholders

There are several actors and agents interested or affected by this project.

The author

He will be the main actor of this project. He is responsible for its completion and success of the goals defined.

Thunderbird community

This group is composed mainly by the AMO team. They are responsible for the definition and appliance of the rules that all add-ons must satisfy to appear on the add-on's site listings. Including security, privacy, user experience, and technical aspects.

Thunderbird users

These are the ones that will receive the benefits of the project. They are the ones who will enjoy a better experience and will give us valuable feedback.

Thunderbird project

The Thunderbird project benefits from a strong add-on community. It makes the application more attractive and powerful for the users.

3.2. Glossary

Add-on	A software component that adds a specific feature to an existing computer program. Also called “plug-in”, “add-on”, or “extension”.
Thunderbird	Email client application produced by Mozilla and their community.
Spell checker	Software component that checks a text against a language dictionary and a set of rules, and returns a list of errors and suggestions to fix each one of them.

3.3. Requirements

As I was using a lot the email back then, I wanted to reduce the effort that implied writing emails. And one of them was this little task of choosing the right language on the spell-checker. I asked my coworkers and they agreed that it could be useful. It was something they did not expect Thunderbird to do, but once you get used to it, it could save time.

3.3.1. Functional requirements

We describe the functional requirements by iteration because we were not going to implement all the features at once. This is how extreme programming works. You choose which user stories will be supported in each iteration.

3.3.1.1. Iteration 1

In this first iteration our main target is to implement a basic working prototype to test mainly if it solves the problem or not, and also to check that it's possible to implement.

To do so we have to dive into Thunderbird code base and Mozilla documentation to know how to contribute, among other tasks.

Requirement	**Define a language for a recipient**
Description:	User is able to define a dictionary language associated to a recipient.
Rationale:	System needs to know the language of a recipient to use it later.
Originator:	Author

Requirement	**Set dictionary language based on recipients**
Description:	System detects the recipient the user is writing the email and changes the dictionary to his language if it was previously defined.
Rationale:	User wants to have the right language set on the spell-checker.
Originator:	Author

3.3.1.2. Iteration 2

This iteration has no new functional requirements.

In this phase we will:
- Fix the case when user has two compose windows opened at the same time.

Also will include quality requirements that had been postponed until now:
- Localization
- Certification

3.3.1.3. Iteration 3

This iteration has no new functional requirements.

New non functional requirements introduced in this iteration are:

1. Localization in Spanish and Catalan

   To have a potentially bigger user-base we decide to translate the plug-in to Spanish and Catalan. We chose those languages because they are the ones I know.

2. Speed and efficiency

   The add-on as is on version 1.0.1 has no control over how much data is stored in RecipientsLanguageDictionary. The data stored will grow indefinitely over time. This can become a problem of space consumed in disk and time needed to load and store this data.

3. Upgradeability

   As we have real users using the add-on, on each release we have to take into account that there are users on previous versions that can upgrade and we have to prepare the plug-in for that. To not lose data between upgrades.

3.3.1.4. Iteration 4

Following our road map we have the epic of:

“Implement more sophisticated logic of remembering.”

This could be described as:

1. We can use different languages depending on the group of recipients more than on the first recipient itself. This is why we will store the language for all the group, not only for individuals.

2. Once we store the language for a group, we store the language for the individuals only if they do not have a previous language stored.

Given that we are limiting the number of pairs stored in our RecipientLanguageDictionary, we need to make sure to not pollute it with useless data. This is why we change the behavior to:

1. Discard a RecipientConfiguration when the number of recipients goes over a threshold.

2. This max number of recipients will be stored as a preference and is editable by the user.

Functional requirements

Requirement	**Define a language for a recipient (version 2)**
Description:	User is able to define a dictionary language associated to a recipient combination. It defines the language for each recipient alone only if they do not already have a language defined.
Rationale:	System needs to know the language of a recipient combination to use it later. It needs also to remember languages for individuals in case we write to them alone in the future.
Originator:	Author

Requirement	**Set dictionary language based on recipients (version 2)**
Description:	System detects which recipient combination is the user writing an email to and changes the dictionary to their language if it was previously defined. If they did not have a language as a recipient combination, it searches for language preferences for each recipient individually.

Rationale:	User wants to have the right language set on the spell-checker.
Originator:	Author

Requirement	**Set storage maximum size**
Description:	User is able to set a max number of language preferences stored. System will limit the number of recipients languages stored to that preference value.
Rationale:	Let user decide how much space he want to use.
Originator:	Author

Requirement	**Set maximum recipients combination size**
Description:	User is able to set the maximum recipients combination size the plug-in will manage. When a recipient combination goes over this limit, the add-on will discard those recipients.
Rationale:	Let user decide which size of the recipients combination is too much.
Originator:	Author

3.3.1.5. Iteration 5

Following our road map we have the epic of:

> “Collect usage data to have global feedback.”

As this epic is not useful for the users, and we received a request to implement it from a user, we decided to add the heuristic implementation to the list. The epic is:

> “Implement heuristics to guess when we have no saved language for a recipient.”

Collecting data

We need to know properties of the users like:

- Language he uses.
- Language he writes to.
- Does the plug-in suggest the right language?

Basically this data.

Heuristic

Based on our problem, an easy way to try to guess which language will have an recipient is to save statistics about the recipient properties and try to guess on the new ones based on history.

In this case, we decide that a good property can be the domain part of the email.

We keep statistics of the probability of each language on each domain and TLD.

An example. If my add-on preferences are:

- alice@foo.it -> “Italian”
- bob@foo.it -> “Italian”

If I write an email to the unknown recipient “john@foo.it”, the add-on will say “Italian”, because the Italian is the more frequent for the “.it” suffix.

From these new features we extract the following requirements:

Functional requirements

Requirement	**Set dictionary language based on recipients (version 3)**
Description:	System detects which recipient combination is the user writing an email to and changes the dictionary to their language if it was previously defined. If they did not have a language as a recipient combination, it searches for language preferences for each recipient individually. If none of them have a defined language, the system tries to guess the language of those recipients using heuristics.
Rationale:	User wants to have the right language set on the spell-checker.
Originator:	Author and user

Requirement	**Set if he wants to send anonymous data**
Description:	User is able to set the preference “send anonymous data”. System will not send statistical data unless it is enabled.
Rationale:	Let user decide if he wants to share anonymous usage data.
Originator:	Author

Requirement	**See anonymous usage data**
Description:	Developer is able to see anonymous usage data of the plug-in.
Rationale:	Collect data from users to take informed decisions in the future.
Originator:	Author

3.3.1.6. Iteration 6

Following our road map we have the epic of:

> “Ask users for reviews”

Like we did on the last release, we'll add something that users want together with this.

In this case, as we received the request from Giacomo in previous iteration (see 7.5.4 Community feedback) asking for Conversations add-on support, we decide to add this to the targets of this release.

Also, Mozilla is pushing developers to convert their add-ons to be “start-less”. This means that users does not have to restart Thunderbird to use the add-on. We add this feature also to make it more attractive and increase users installation.

The targets are:

- Ask users for reviews.
- Compatibility with Conversations add-on.
- Restart-less.

Requirement	**Ask users for reviews**
Description:	Add-on asks user to write a review from time to time.
Rationale:	Get more feedback from the users, to gain more installs that would potentially increase the user base.
Originator:	Author

Quality Requirement	**Compatibility with Conversations plug-in**
Description:	Allow users to use the add-on together with Conversations
Rationale:	Some users want to use these add-ons together.
Originator:	User

Quality Requirement	**Add-on is start-less**
Description:	Users can install it without restarting Thunderbird
Rationale:	Add-on will be easier to use and gets highlighted on the AMO listings, which can lead to more users.
Originator:	Mozilla

3.3.1.7. Iteration 7

We have no pending target to implement from our road map, but we have blocking issues from some Windows users that we should take care of, so the target of this iteration will be fixing bugs.

This iteration does not add any requirement, only focuses on the already existing requirement of being compatible with all operating systems that are compatible with Thunderbird.

3.3.2. Quality requirements

3.3.2.1. Look and feel

Requirement	**Add-on look and feel**
Description:	The communication to the user has to have the same appearance as the rest of the Thunderbird application
Rationale:	We want the add-on to seem as part of the application
Originator:	Author

3.3.2.2. Usability and human requirements

Requirement	**Simplicity**
Description:	The usage of the add-on has to be simple.
Rationale:	It has to be usable by all Thunderbird users, without expecting technical skills.
Originator:	Author

Requirement	**Language**
Description:	The add-on and its documentation must be in English.
Rationale:	It's the language of the Thunderbird community.
Originator:	Author

Requirement	**Localization**
Description:	The add-on interaction and its documentation can be localized.
Rationale:	Thunderbird users use a lot of different languages and not all understand English, so it's good to speak their language.
Originator:	Author

Requirement	**Minimal interaction**
Description:	The add-on will interact with the user only when it needs human interaction or has to inform about some action taken.
Rationale:	We do not want to bother the user. The tool has to remove work, not add more.
Originator:	Author

3.3.2.3. Performance requirements

Requirement	**Speed and efficiency**
Description:	The add-on should not block application usage and resources.
Rationale:	The add-on cannot block the user from writing or composing. It has to work on the background.
Originator:	Author

3.3.2.4. Security requirements

Requirement	**Privacy**
Description:	Add-on must not share user information between Thunderbird accounts.
Rationale:	Data must be stored somewhere only accessible by the user.
Originator:	Author

3.3.2.5. Other requirements

Requirement	**Operating platforms**
Description:	The add-on will be supported in any platform that supports Thunderbird
Rationale:	To not reduce the user base to a subset of it.
Originator:	Author

Requirement	**Thunderbird compatibility**
Description:	The add-on must operate with Thunderbird defined interface.
Rationale:	As this plug-in has to run inside Thunderbird, we have to use its internal components interfaces.
Originator:	Author

Requirement	**Certification**
Description:	The add-on must be accepted by Thunderbird Add-ons team.
Rationale:	To be listed in their add-ons directory, we must comply their rules.
Originator:	Author

Requirement	**Deployment**
Description:	The add-on has to be deployed and updated by Thunderbird add-ons installer
Rationale:	When user install this add-on, it's a task done by Thunderbird itself, so we need to be compatible with this process and create an installable package.
Originator:	Author

Requirement	**Upgradeability**
Description:	The add-on can be installed on systems where a previous version of the add-on existed and do not remove previous data or preferences.
Rationale:	When user upgrades the add-on, we need to migrate data structures and preferences in case they changed. We can't count on a fresh install nor lose data either.
Originator:	Author

Requirement	**Open source**
Description:	All the source code has to be public.
Rationale:	To allow developers to contribute to it or do their own versions. And because there is no reason to not be open.
Originator:	Author

Requirement	**For free**
Description:	User never has to pay or sign any contract for it.
Rationale:	We want to contribute to the community, it's altruistic.
Originator:	Author

4. Specification

4.1. Actors

We have one type of actor in our system, it's the User. But we also have Thunderbird as an actor because some actions are triggered by the platform itself.

User

The user that uses Thunderbird and has the add-on installed and active.

Thunderbird

The Thunderbird application can trigger events so it's an actor on this system.

4.2. Use cases

The use cases that we are describing already existed in the application, but we re-describe them because we are changing their behavior.

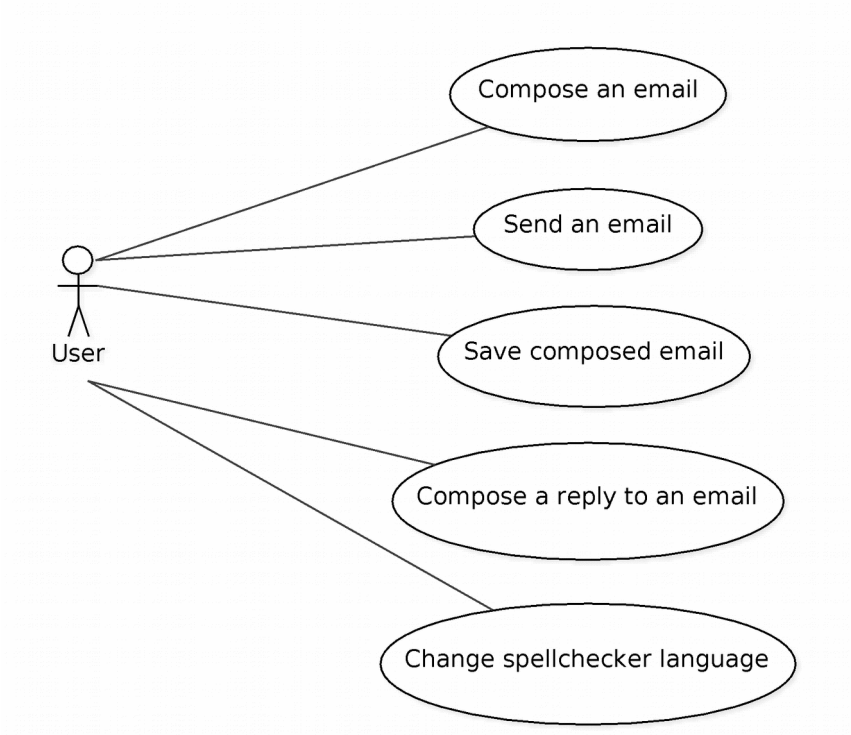


Illustration 6: Use cases

4.2.1. Iteration 1

Use case Compose an email

Primary actor: User

Preconditions:
- User exists on the platform

Postconditions:
- A new email exist on the system

Main scenario:
1. The user tells that he wants to compose a new email.
2. The system shows the form with recipients, subject and body.
3. The user introduces the recipients.
4. The system detects the recipients and sets spell-checker language for the first recipient that has a stored language preference.
5. The user writes the subject and the message

Extensions:
1. In step 4: If it doesn't have a language stored for them, it does nothing and follows step 5.
2. In any step: User can cancel the use case.
3. In any step: User can start the use case “change spell-checker language”.

Use case Change spell-checker language

Primary actor: User

Preconditions:

Postconditions:
- Spell checker language has changed

Main scenario:
1. The user tells that he wants to change the spell-checker language.
2. The system shows a list of languages available
3. The user selects a language
4. The system changes spell-checker current language to the one selected.
5. The system registers the language for each recipient of the email.

Extensions:
1. In any step: User can cancel the use case.
2. In step 5: If the recipients list is empty, ends use case.

Use case Send an email

Primary actor: User

Preconditions:
- The email exists in the system

Postconditions:
- The email has been sent

Main scenario:
1. The user tells that he wants to send the email
2. System sends the email

Extensions:
1. In step 2: Sending email fails. Notify user and end use case.

Use case Save composed email

Primary actor: User

Preconditions:

Postconditions:
- The email has been saved as draft

Main scenario:
1. The user tells that he wants to save the composed email
2. System saves the email content.

Use case Compose a reply to an email

Primary actor: User

Preconditions:
- The email to reply exists on the system

Postconditions:
- The reply has been composed

Main scenario:
1. The user tells that he wants to compose a reply of an email
2. The system shows the form with the recipients of the original email, subject already filled and an empty body.
3. The system sets spell-checker language for the first recipient of the recipients list that has a language preference.
4. The user writes the body

Extensions:
1. On step 3: if the system does not have a previously registered recipient language for any of the recipients, it does not change the spell-checker language.
2. On step 4: user can start the use case “Change spell-checker language” and go back to step 4 after that.
3. In any step, user can cancel the use case.

4.2.2. Iteration 2

In this iteration there is no new use case to cover.

4.2.3. Iteration 3

In this iteration there is no new case to cover.

4.2.4. Iteration 4

In this iteration we modify 3 previous use cases and add 2 more.

Use case Compose an email (Version 2)

Primary actor: User

Preconditions:
- User exists on the platform

Postconditions:
- A new email exist on the system

Main scenario:
1. The user tells that he wants to compose a new email.
2. The system shows the form with recipients, subject and body.
3. The user introduces the recipients.
4. The system detects the recipients change and sets the spell-checker language for the language associated to that recipient combination. If there is no previous stored language for them, it searches for each recipient individually and sets the first language it finds.
5. The user writes the subject and the message.

Extensions:
1. In step 4: If it has no language stored for them, it does nothing and follows step 5.
2. In any step: User can cancel the use case.
3. In any step: User can start the use case “change spell-checker language”.

Use case Change spell-checker language (version 2)

Primary actor: User

Preconditions:

Postconditions:
- Spell checker language has changed

Main scenario:
1. The user tells that he wants to change the spell-checker language.
2. The system shows a list of languages available.
3. The user selects a language.
4. The system changes spell-checker current language to the one selected.
5. The system registers the language for the recipient combination as a whole and for each recipient individually only if they had no one defined.

Extensions:
1. In any step: User can cancel the use case.
2. In step 5: If the recipients list is empty, ends use case.

Use case Compose a reply to an email (version 2)

Primary actor: User

Preconditions:
- The email to reply exists on the system

Postconditions:
- The reply has been composed

Main scenario:
1. The user tells that he wants to compose a reply of an email.
2. The system shows the form with the recipients of the original email, subject already filled and an empty body.
3. The system sets the spell-checker language to the one we had stored for the recipient combination. If there is no preference stored, it searches for each recipient and sets the first it finds.
4. The user writes the body.

Extensions:
1. On step 3: if the system does not have a previously registered language preference for any of the recipients, it does not change the spell-checker language. And it follows on step 4.
2. On step 4: user can start the use case “Change spell-checker language” and go back to step 4 after that.
3. In any step, user can cancel the use case.

Use case Set storage maximum size preference

Primary actor: User

Preconditions:

Postconditions:
- The preference value for max size has changed

Main scenario:
1. The user tells that he wants to change the preference max storage size.
2. The system shows a form with the input to change it's value.
3. User edits the value and saves the form.
4. The system sets the value to the value in the form.

Extensions:
1. In any step, user can cancel the use case.
2. In step 4: if the introduced value is not a number or positive, it will show a message to the user and go to step 2.

Use case	Set maximum recipients combination size
Primary actor:	User
Preconditions:	
Postconditions:	• The preference value for max recipients combination size has changed.

Main scenario:

1. The user tells that he wants to change the preference for maximum recipients combination size.
2. The system shows a form with the input to change it's value.
3. User edits the value and saves the form.
4. The system sets the preference value to the value in the form.

Extensions:

1. In any step, user can cancel the use case.
2. In step 4: if value is not a number or positive, it will show a message to the user and go to step 2.

4.2.5. Iteration 5

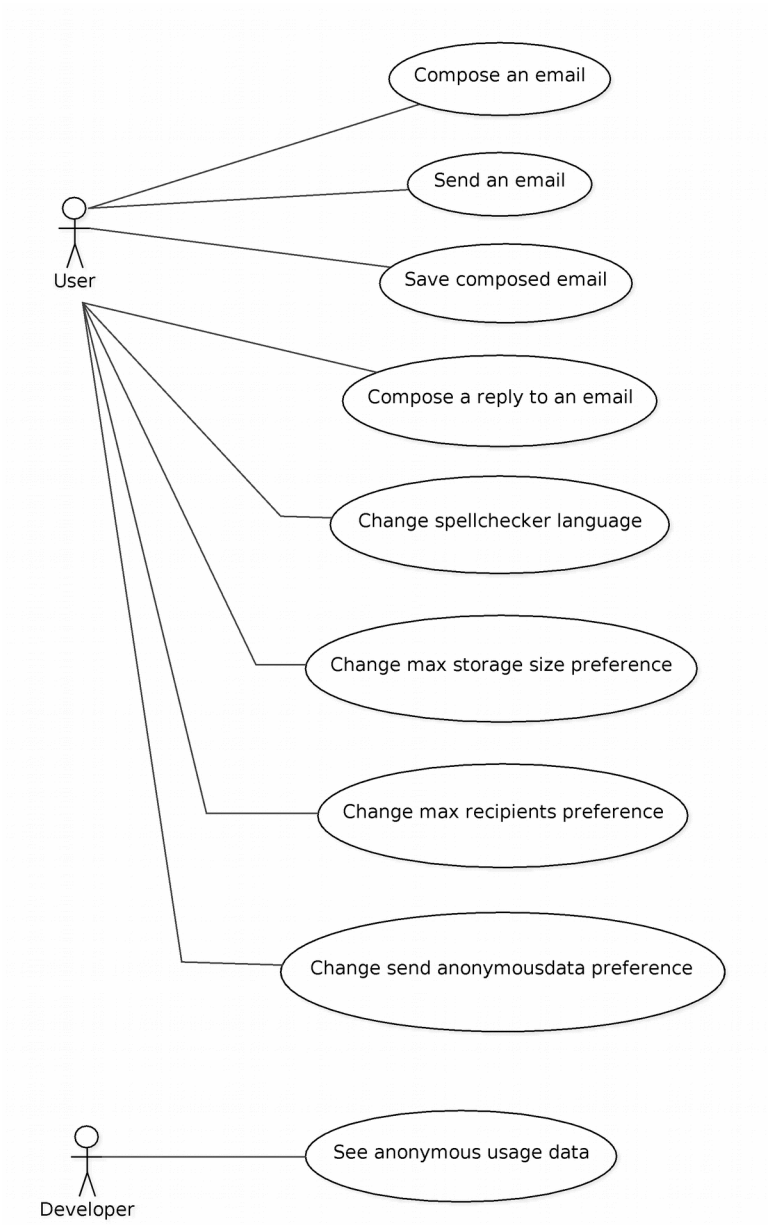


Illustration 7: Use cases on iteration 5

In this iteration we modify 3 previous use cases and add 2 more.

Use case Compose an email (Version 3)

Primary actor: User

Preconditions:
- User exists on the platform

Postconditions:
- A new email exist on the system

Main scenario:
1. The user tells that he wants to compose a new email.
2. The system shows the form with recipients, subject and body.
3. The user introduces the recipients.
4. The system detects the recipients change and sets the spell-checker language for the language associated to that recipient combination. If there is no previous stored language for them, it searches for each recipient individually and sets the first language it finds.
   If it can't find a language either, it uses heuristics to guess the language.
5. The user writes the subject and the message

Extensions:
1. In step 4: If it does not know any recipient or has no language stored for them, or heuristic does not work, it does nothing and follows step 5.
2. In any step: User can cancel the use case.
3. In any step: User can start the use case “change spell-checker language”.

Use case Change spell-checker language (version 3)

Primary actor: User

Preconditions:

Postconditions:
- Spell checker language has changed

Main scenario:
1. The user tells that he wants to change the spell-checker language.
2. The system shows a list of languages available
3. The user selects the desired language
4. The system changes spell-checker current language to the one selected.
5. The system registers the language for the recipient combination as a whole and for each recipient individually only if they had no one defined. The system also updates heuristic data from that change.

Extensions:
1. In any step: User can cancel the use case.
2. In step 5: If the recipients list is empty, ends use case.

Use case	Compose a reply to an email (version 3)
Primary actor:	User
Preconditions:	• The email to reply exists on the system
Postconditions:	• The reply has been composed

Main scenario:

1. The user tells that he wants to compose a reply of an email
2. The system shows the form with the recipients of the original email, subject already filled and an empty body.
3. The system sets spell-checker language that we had stored for the recipient combination. If there is no preference stored, it searches for each recipient and sets the first preference it finds. If it can't find a language either, it uses heuristic to guess the new language.
4. The user writes the body.

Extensions:

1. On step 3: if the system does not have a previously registered language preference for any of the recipients, or heuristic does not work, it does not change the spell-checker language. And follows on step 4.
2. On step 4: user can start the use case “Change spell-checker language” and go back to step 4 after that.
3. In any step, user can cancel the use case.

Use case	Set preference on sending anonymous statistic data
Primary actor:	User
Preconditions:	
Postconditions:	• The preference for sending anonymous data has changed.

Main scenario:

1. The user tells that he wants to change the preference “send anonymous data”.
2. The system shows a form with the input to change it's value.
3. User edits the value and saves the form.
4. The system sets the preference value to the value in the form.

Extensions:

1. In any step, user can cancel the use case.

Use case	See anonymous usage data
Primary actor:	Developer
Preconditions:	
Postconditions:	

Main scenario:

1. The developer asks to see statistical usage data to the system.
2. The system shows a report of the plug-in usage.

Extensions:

1. In any step, user can cancel the use case.

4.2.6. Iteration 6

In this iteration we have a new use case:

Use case	Ask for reviews
Primary actor:	System
Preconditions:	• User is composing a message
Postconditions:	

Main scenario:

1. The system detects that the user has been using the add-on for some time and asks the user for a review. It shows a message to the user with information on how to proceed.
2. The user follows the link and completes the process of writing a review.
3. The system stores the information that the user has been asked now, to not ask him after some time.

Extensions:

1. In step 2: user can say no to write a review. End of use case.

4.2.7. Iteration 7

This iteration has no changes on defined use cases.

4.3. Conceptual model

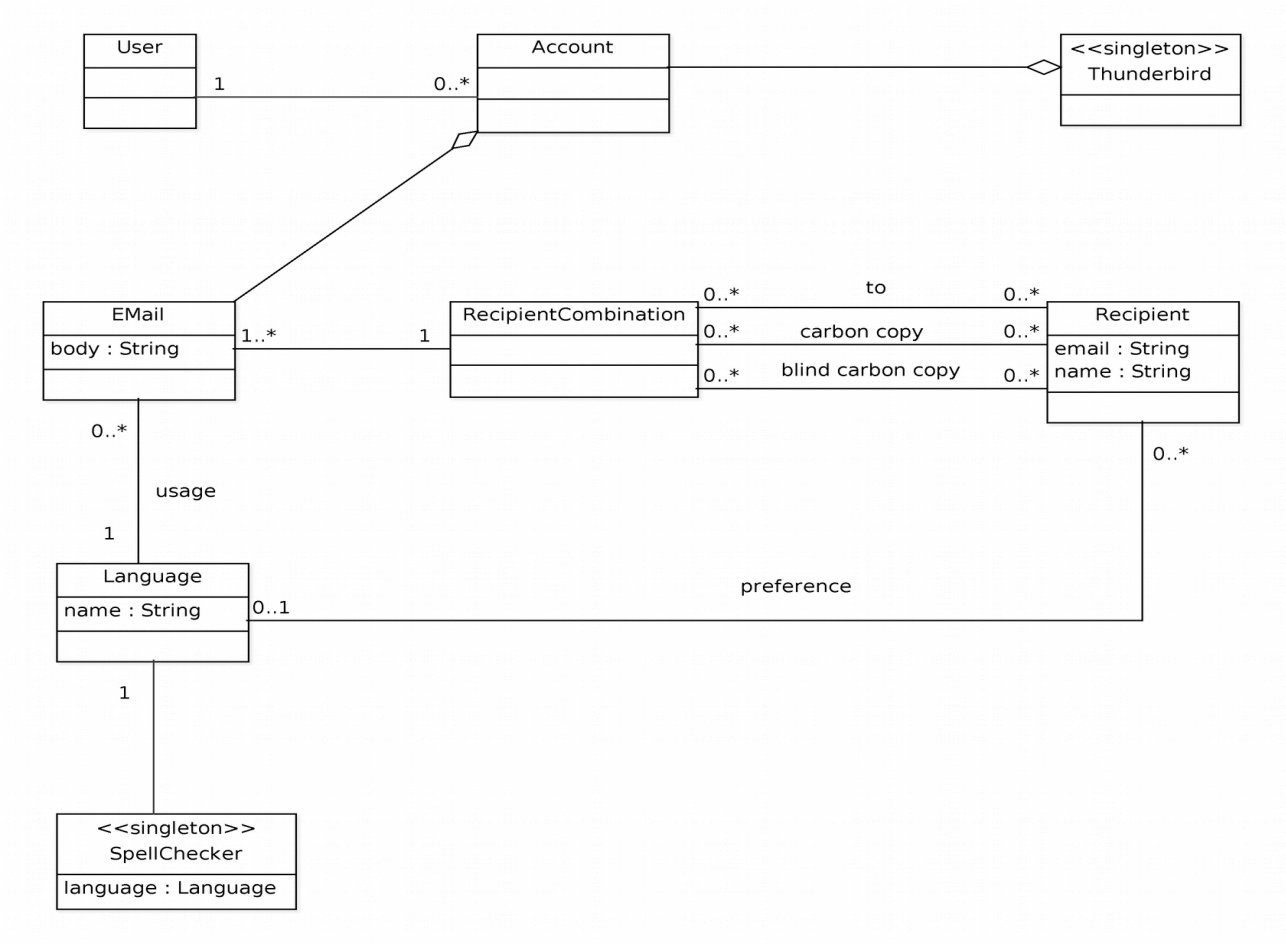


Illustration 8: Conceptual model

Constraints

- Language names must be unique.
- Recipient emails must be unique.
- Recipient Combination relations with Recipient (to, carbon copy and blind carbon copy) have to be unique, without taking order into account. They could be understood as 3 sets of Recipients.

User

A user of Thunderbird that uses the add-on. It can have several accounts.

Account

Is the email account of a user. This account is configured in Thunderbird with all the data required to read emails form the server and send them.

Email

Is the content that is sent from an email account to another. It has a Language usage because we use a language when writing it.

- **body**: A text with the message that the email will be transmitting. This is usually text but can be images or other files.

Language

The method of human communication, either spoken or written, consisting of the use of words in a structured and conventional way.

Recipient Combination

Its the combination of recipients in an email. This combination is a collection of three lists. The ones the email is targeted, the ones which will receive a copy (known as “Carbon Copy”), and the ones who receive a secret copy (known as “Blind Carbon Copy”) without the others knowing about it.

Recipient

It's the target of an email. Represents the destination that will receive this email if it's finally sent.

- **email**: A string with the email in its standardized format. This is the primary key.
- **name**: The name of the recipient. Recipients can have names besides their email.

Thunderbird

The email client application with its add-ons.

4.3.1. Iteration 4

We update the conceptual model in this iteration with the preferences object and the relation between RecipientCombination and Language.

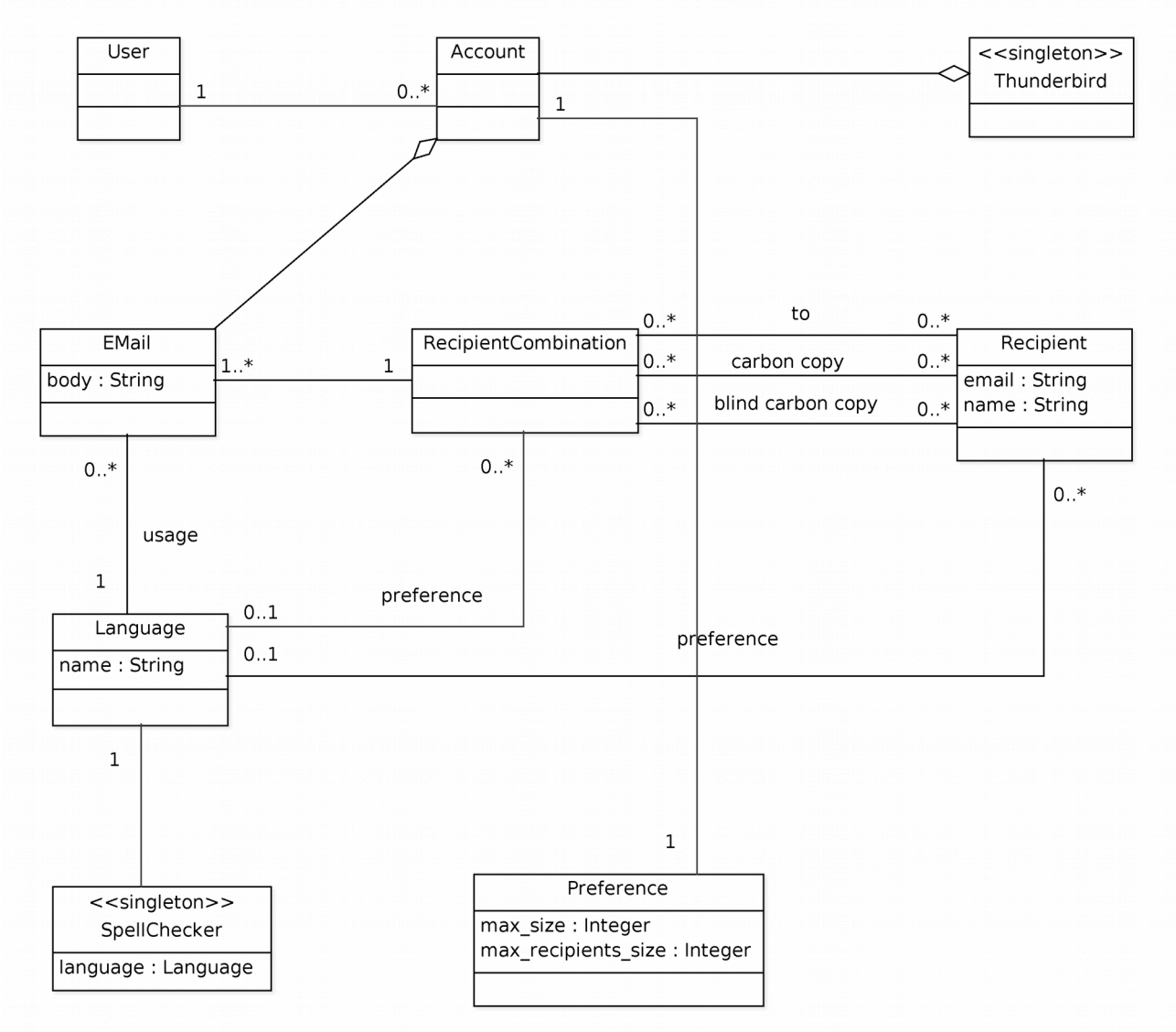


Illustration 9: Conceptual model on iteration 4

The constraints are the same as in the original but adding these:

1. Preference max_size is a positive integer.
2. Preference max_recipients_size is a positive integer.

Preference

Is the model where we store global preferences for each account.

4.3.2. Iteration 5

We update the conceptual model in this iteration with a new preference and Google Analytics model.

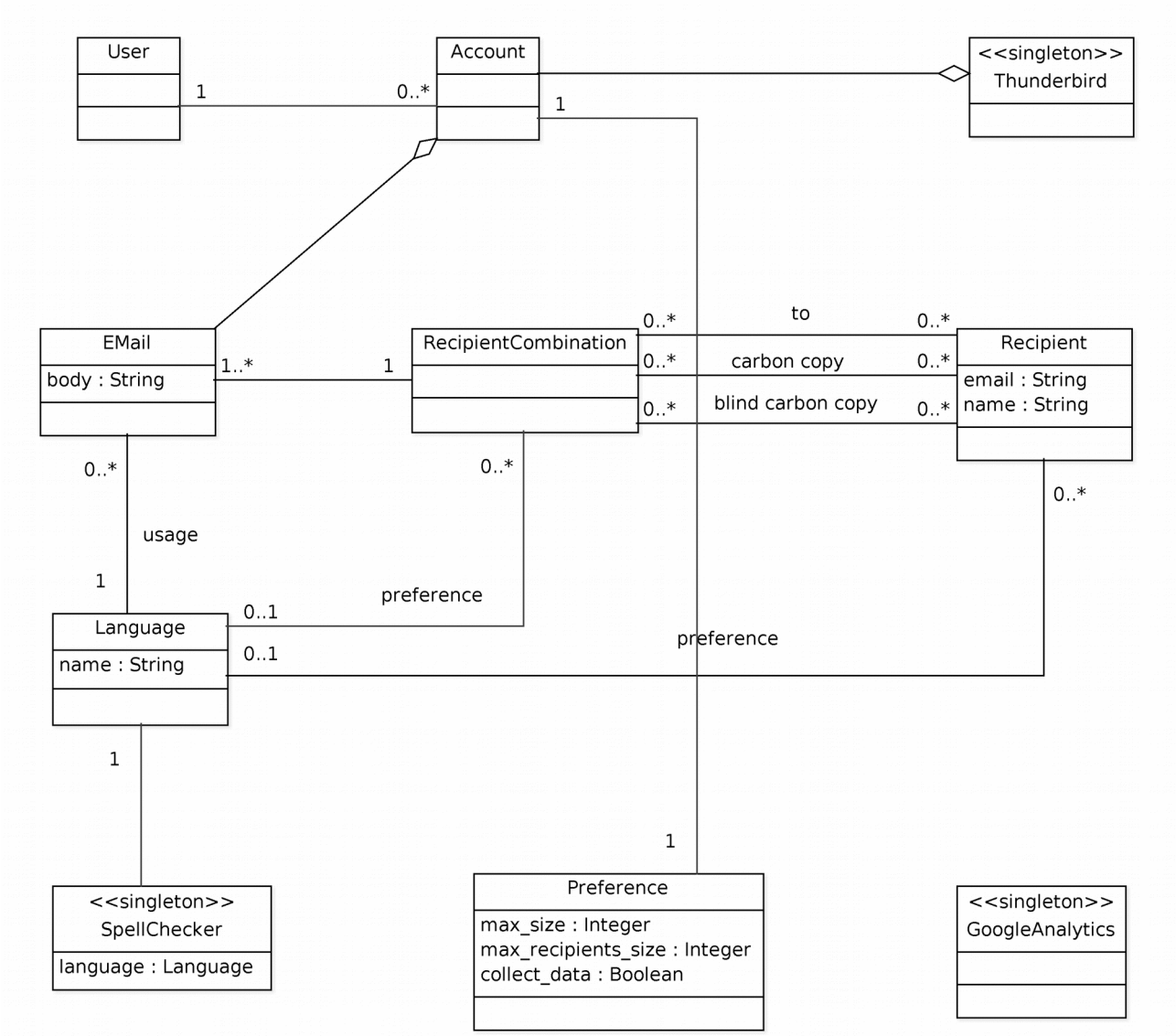


Illustration 10: Conceptual model on iteration 5

Google Analytics

Represents the external service where we send the anonymous data.

4.4. Behavior model

4.4.1. Iteration 1

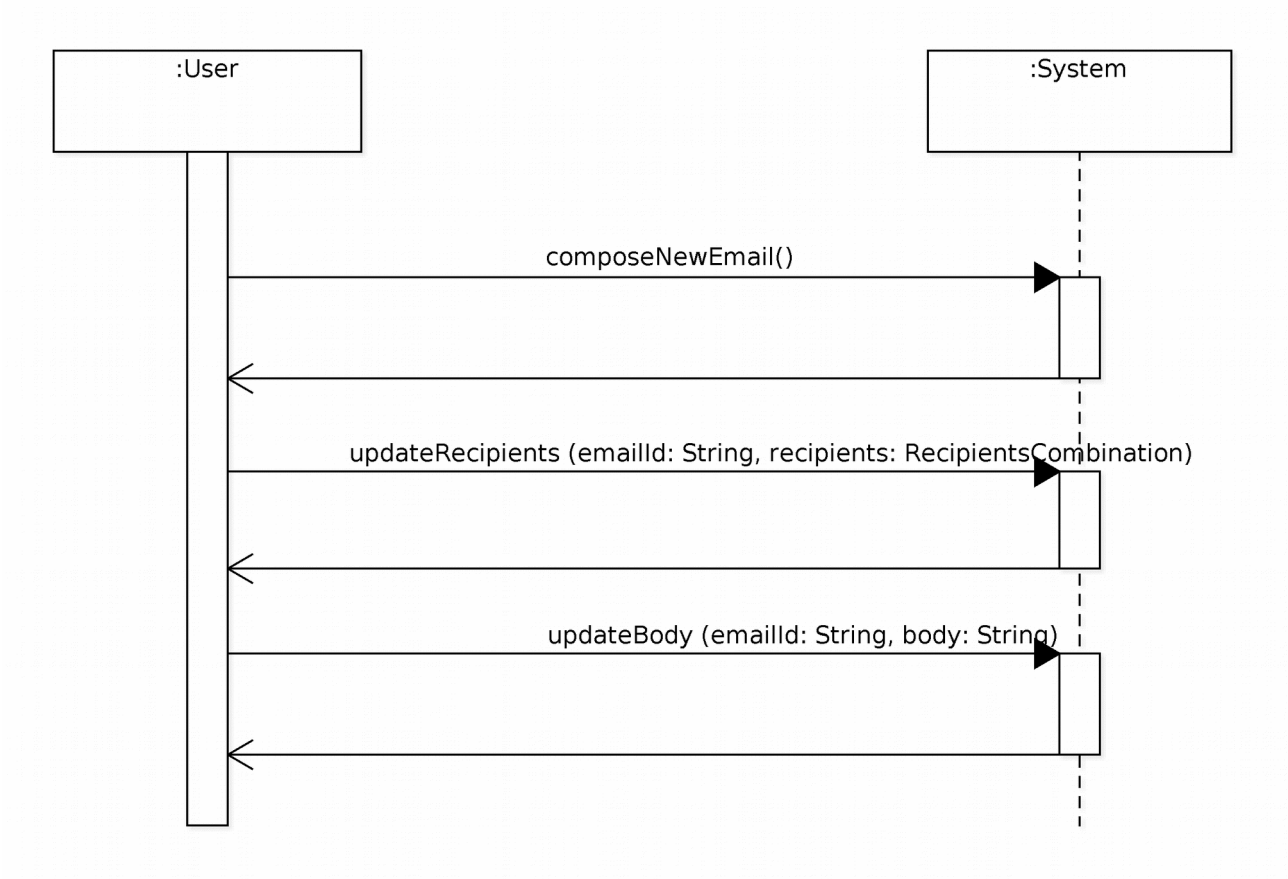


Illustration 11: Compose new email sequence diagram

context	**composeNewEmail(): Email**
Pre:	(none)
Post:	Creates an Email in the system with empty recipients and body

context	**updateRecipients(emailId: String, recipients: RecipientCombination)**
Pre:	Email with emailId exists
Post:	Email recipients are the recipients provided. Spell-checker language is changed to the one assigned to the first recipient in recipients list that has a language defined. No change if no recipient has a defined language.

context **updateBody(emailId: String, body: String)**

Pre: Email with emailId exists

Post: Email body is the String body

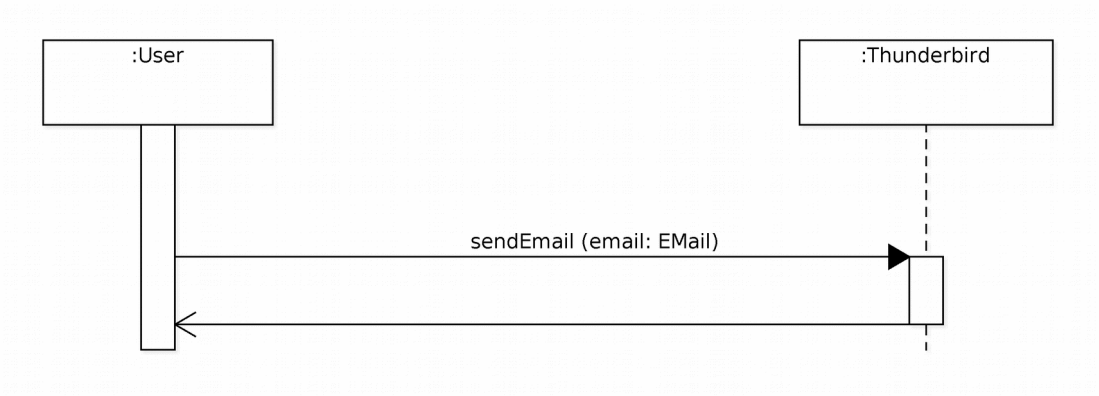


Illustration 12: sendEmail sequence diagram

context **sendEmail(emailId: String)**

Pre: Email with emailId exists

Post: System sends the email to the recipients of the email.

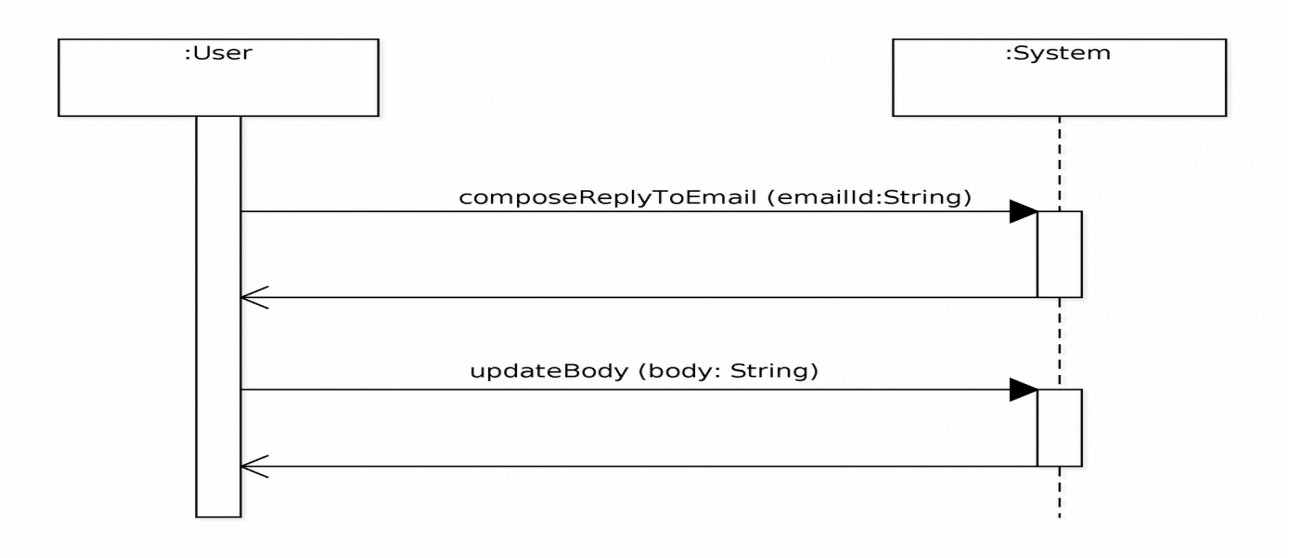


Illustration 13: composeReplyToEmail sequence diagram

context **composeReplyToEmail(emailId: String): String**

Pre: Email with emailId exists

Post: System creates a new Email with the same recipients, predefined subject and empty body.
System sets spell checker language to the first recipient in the email that has a language predefined. It does not change the spell-checker language if none of the recipients have a predefined language.

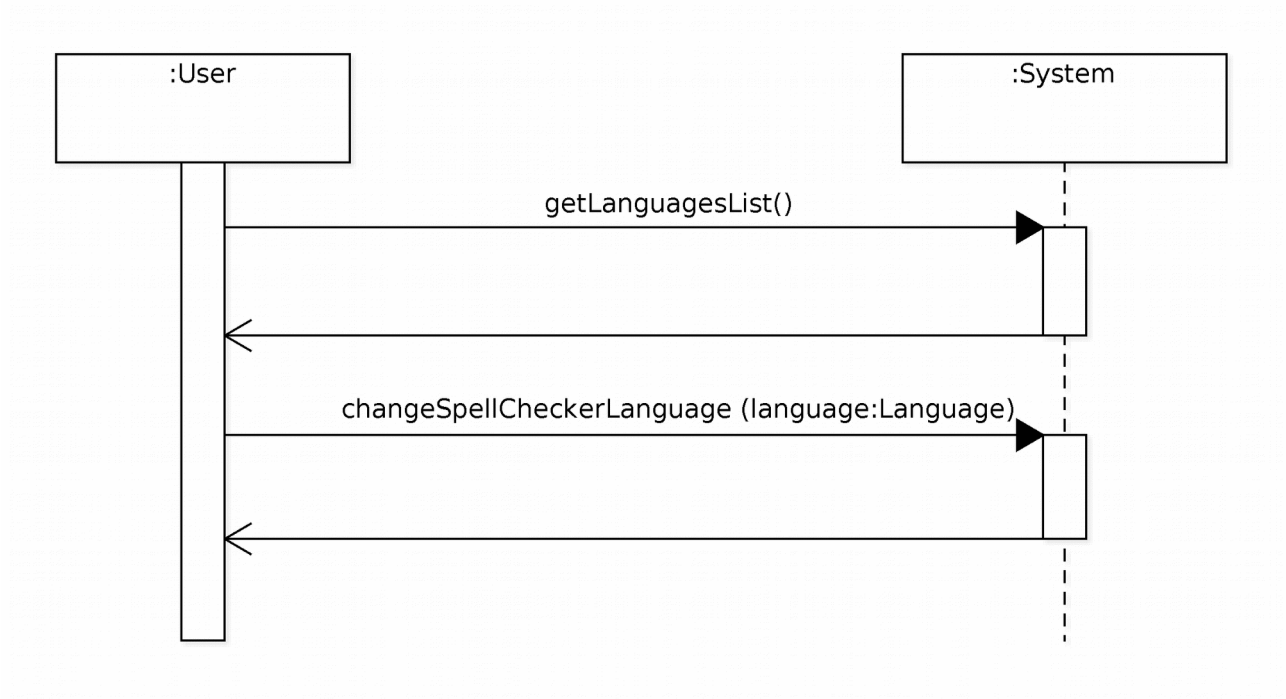


Illustration 14: changeSpellCheckerLanguage sequence diagram

context	**getLanguagesList()**
Pre:	(none)
Post:	Returns a list of all the languages available.

context	**changeSpellCheckerLanguage(languageId: Language)**
Pre:	Language with languageId exists
Post:	System changes spell-checker language to languageId. System records language with languageId as preference for all the recipients of the current email.

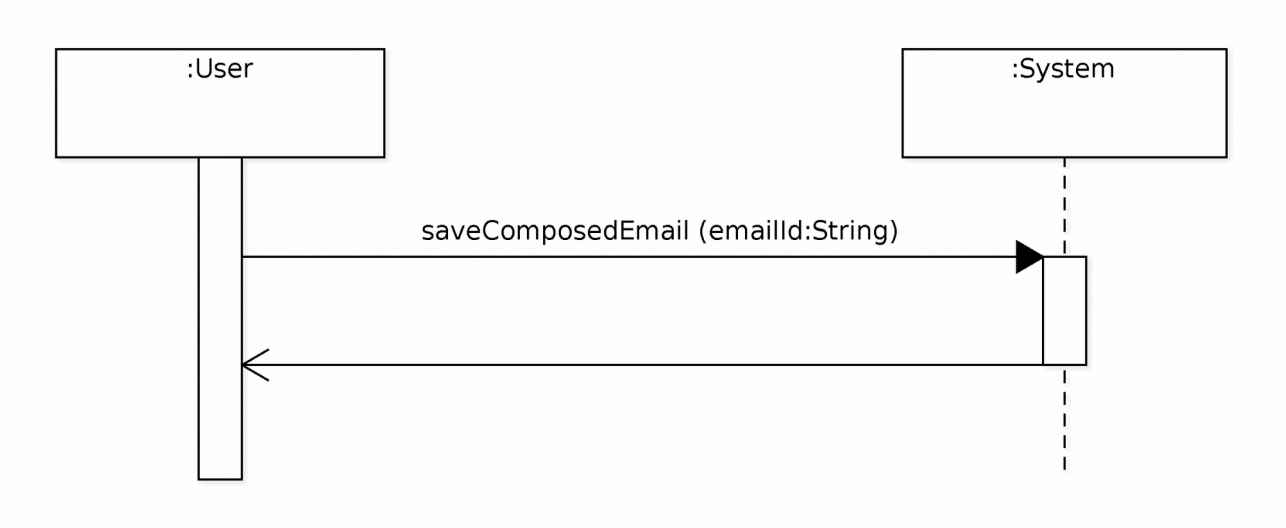


Illustration 15: saveComposedEmail sequence diagram

context	**saveComposedEmail(emailId: String)**
Pre:	Email with emailId exists
Post:	System registers this email as draft.

4.4.2. Iteration 2

In this iteration there is no behavior change.

4.4.3. Iteration 3

In this iteration there is no behavior change.

4.4.4. Iteration 4

context	**updateRecipients(emailId: String, recipients: RecipientCombination)** ***Version 2***
Pre:	Email with emailId exists
Post:	Email recipients are the recipients provided. Spell checker language is changed to the one assigned to the recipient combination. In case it has none defined, we look for the first recipient in recipients that has a language defined. No change if no recipient has a defined language.

context	**composeReplyToEmail(emailId: String): String** **Version 2**
Pre:	Email with emailId exists
Post:	System creates a new Email with the same recipients, predefined subject and empty body. System sets spell-checker language to the language stored for the recipient combination. If it does not exist, it looks for the first recipient in the email that has a language predefined. It does not change the spell checker language if none of the recipients have a predefined language.

context	**changeSpellCheckerLanguage(languageId: Language)** **Version 2**
Pre:	Language with languageId exists
Post:	System changes spell-checker language to languageId. System records language with languageId as preference for the current recipient combination. It also saves the language for the current recipients individually that did not have a language preference before.

4.4.5. Iteration 5

context	**updateRecipients(emailId: String, recipients: RecipientCombination)** ***Version 3***
Pre:	Email with emailId exists
Post:	Email recipients are the recipients provided. Spell checker language is changed to the one assigned to the recipient combination. In case it has none defined, we look for the first recipient in recipients that has a language defined. If no language defined for them, it uses heuristics to guess language. No change if heuristics do not work.

context	**composeReplyToEmail(emailId: String): String** **Version 3**
Pre:	Email with emailId exists
Post:	System creates a new Email with the same recipients, predefined subject and empty body. System sets spell checker language to the language stored for the recipient combination. If it does not exist, it looks for the first recipient in the email that has a language predefined. If none has a preference, it use heuristics to guess the language. It does not change the spell-checker language if none of the previous methods return a candidate language.

context	**changeSpellCheckerLanguage(languageId: Language)** **Version 3**
Pre:	Language with languageId exists
Post:	System changes spell-checker language to languageId. System records language with languageId as preference for the recipient combination. It saves the language for the recipients individually that did not have a language preference before. And updates heuristics stats with these changes.

context	**showUsageData()**
Pre:	Developer exists and is authenticated.
Post:	A system shows plug-in statistical data to the developer.

4.4.6. Iteration 6

context	**askForReview()**

Pre:	(none)
Post:	System shows a message to the user asking kindly for a review.

4.4.7. Iteration 7

In this iteration there is no behavior change.

5. Design

To design this add-on I had to read all the Mozilla on-line documentation about how to develop a plug-in, and understand how the Thunderbird application works. I also had to find out which interfaces I would need to access and how to update or change the application visual interface.

5.1. Architecture

The architecture of the project is conditioned by the architecture defined by Thunderbird.

Thunderbird architecture, based on XUL and Mozilla tools, could be understood as a MVC where:

- Model layer is build with C++ classes that provide persistence and other services access (network, full text search engine, etc.)
- View layer is composed by XUL definitions (graphical objects) together with some JavaScript to build the behavior of the view.
- Controller layer build with C++ and JavaScript, that orchestrates the view and the model layer.

Our plug-in main architecture is “Implicit invocation” because we do not reply to direct user requests but react to some actions he does. We hook on interface or data change events to change application behavior when required.

This means that our main tasks are:

- Hook on certain events to gather changes from the domain
- Interact with the domain to change its behavior.

Also, the Thunderbird plug-in system requires us to:

- Add hooks on code to install, boot, shutdown and uninstall the plug-in, to let Thunderbird call our code. We are not a process or thread ourselves.
- Views have to be in XUL and inserted into the user view dynamically.

5.2. Domain model

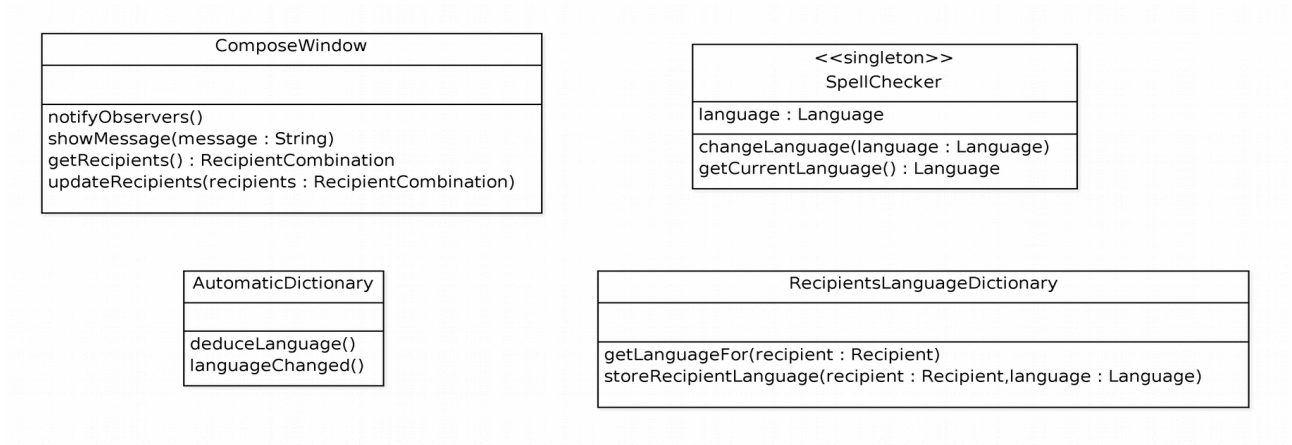


Illustration 16: Domain model

5.2.1. Iteration 3

On this iteration we need to implement dictionary max size and migration logic. The Domain model changes to this:

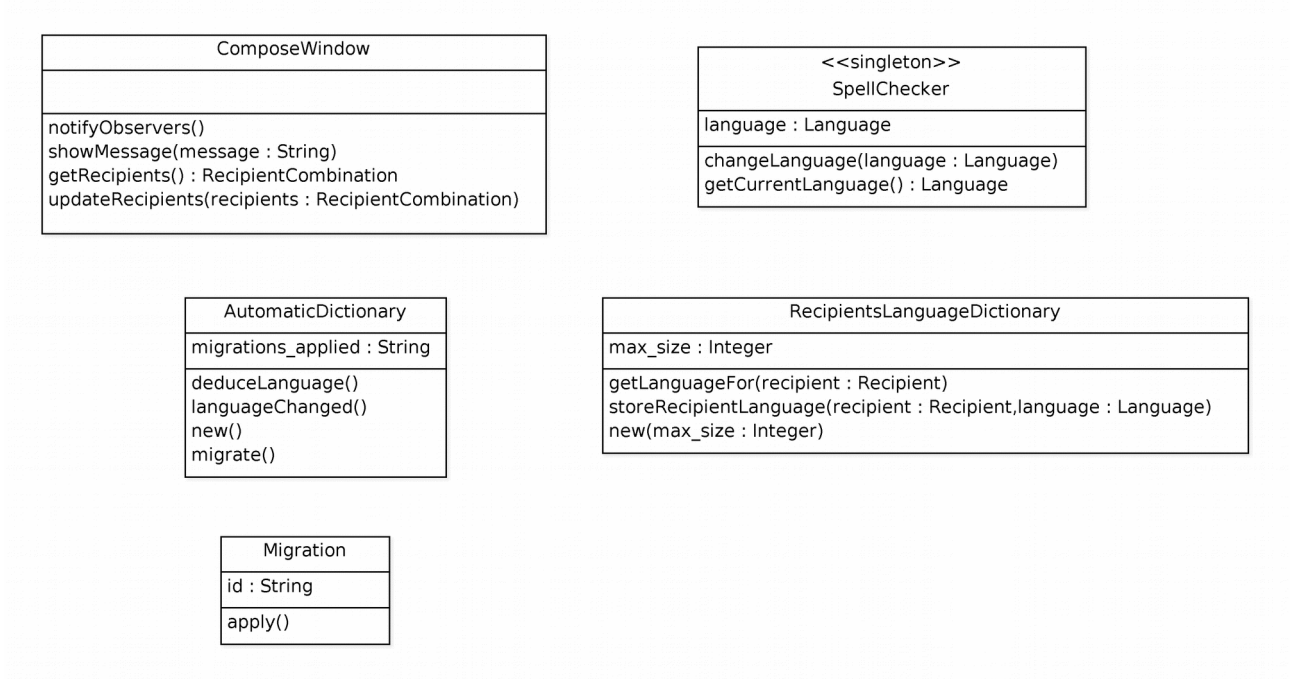


Illustration 17: Domain model on iteration 3

5.2.2. Iteration 4

On this iteration we change the interface of some of the methods of RecipientsLanguageDictionary to accept RecipientsCombination.

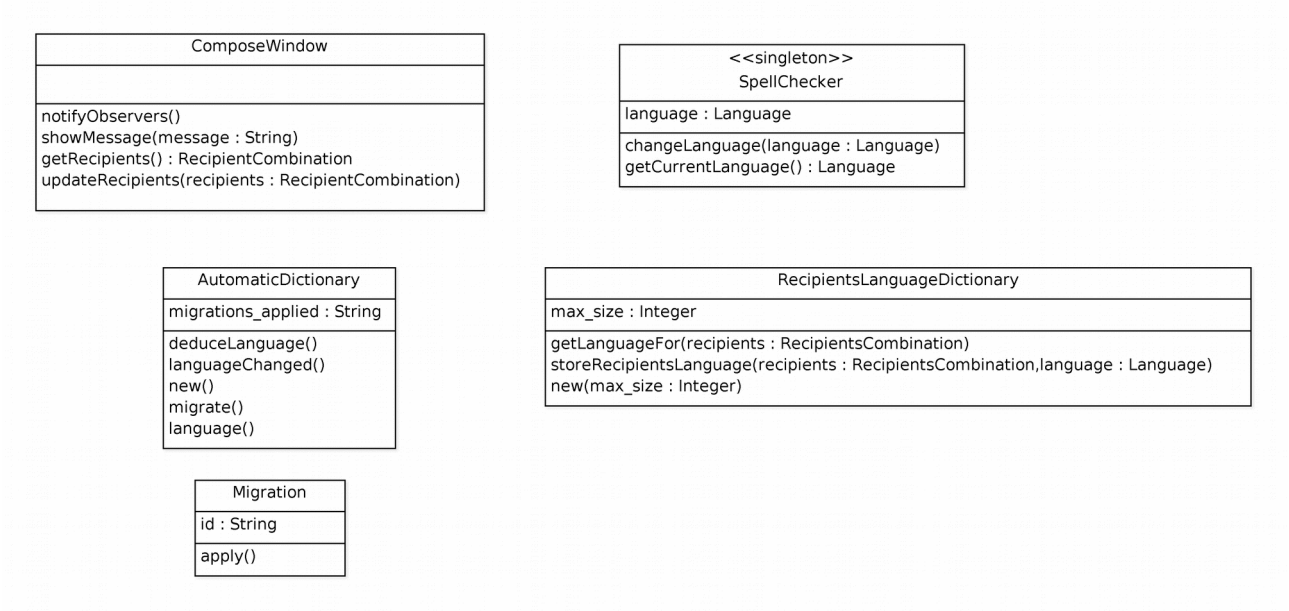


Illustration 18: Domain model on iteration 4

5.2.3. Iteration 5

On this iteration we add the class responsible for the heuristic logic and the class that sends Google Analytics events.

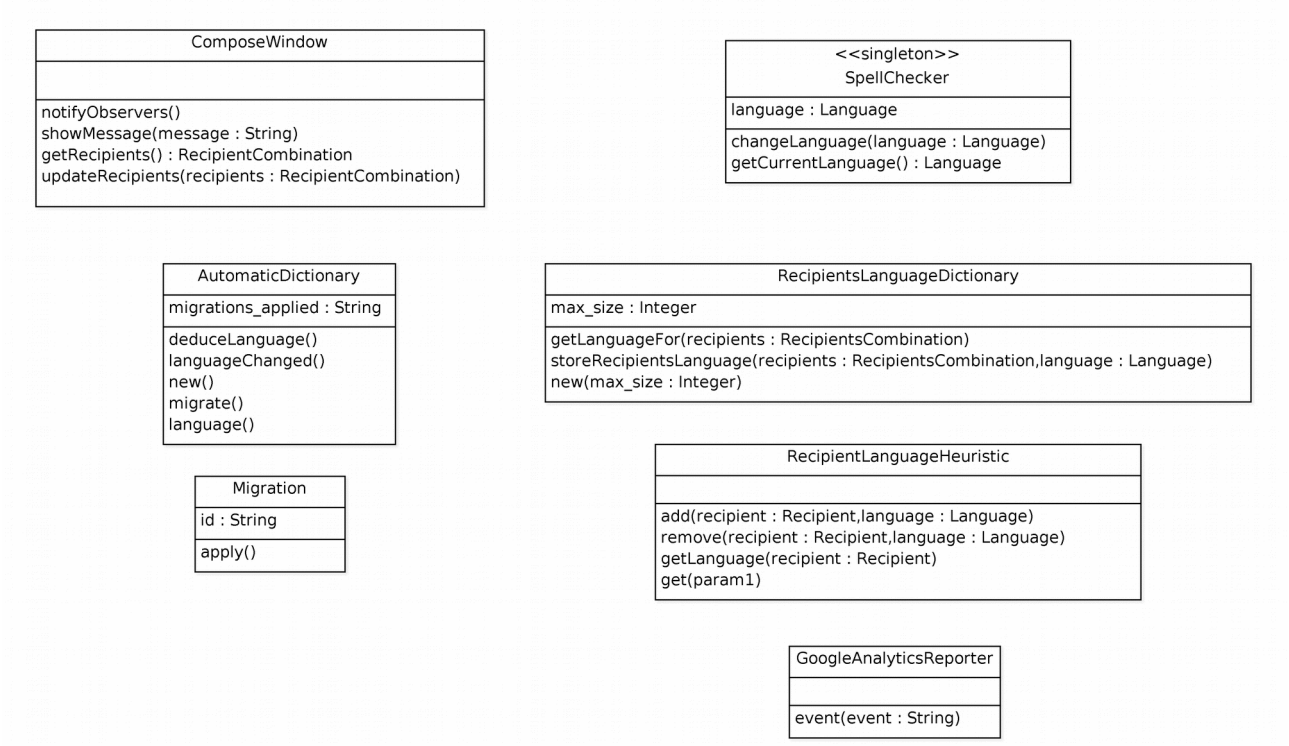


Illustration 19: Domain model on iteration 5

5.2.4. Iteration 6

In this iteration we have the domain model of iteration 5 but adding the method “askUserForReview()” to AutomaticDictionary.

5.2.5. Iteration 7

There is no change in domain model in this iteration.

5.3. Data model

We'll implement RecipientsLanguageDictionary as a simple dictionary with the recipients as keys and languages as values.

As we store this data in Preferences, generating a serialized version of that hash, and read from there every time we need to retrieve data.

This hash keys are the recipient emails, and the values the language identifiers.

5.3.1. Iteration 3

In this iteration we face several changes in the data.

1. The RecipientsLanguageDictionary has a new requirement, the maximum size.

2. To keep upgradeability, we need to implement a process to upgrade data structures from old versions to new ones. We call this process a “data migration”.

RecipientsLanguageDictionary with maximum size

When we add a max_size constraint to the data model, we need to define what will happen when we reach that maximum.

The requisites to implement this are:

– Do not ask user to decide because we can't bother user so much.

– Remove an item that is not going to be used in the future, or is the less likely to.

- Be efficient in time and in space.

Based on this requisites, the chosen algorithm has been “Last Recently Used”. This is a well known cache replacement algorithm that will have O(n) space cost.

We'll create a data migration to migrate user's data from old dictionary to this new version.

5.3.2. Iteration 4

In this iteration we change the interface of the RecipientsLanguageDictionary to accept RecipientsCombination in stead of Recipients alone.

The serialization of the recipients combination needs to work as follows:

- Return the same key even if recipients come in different order.
- It will distinguish recipients between “to” and “cc”.

The result of the serialization will be used in the hash as a key, so it's important to keep those rules to make it work.

5.3.3. Iteration 5

There is a new data, the events we send from the plug-in.

As we want to report user behavior and success rate, we will report the following events:

- Language saved
- Language restored/remembered
- Language guessed
- Execution Errors
- No language found
- Mail sent

– Plug-in boot time

5.4. Sequence diagrams

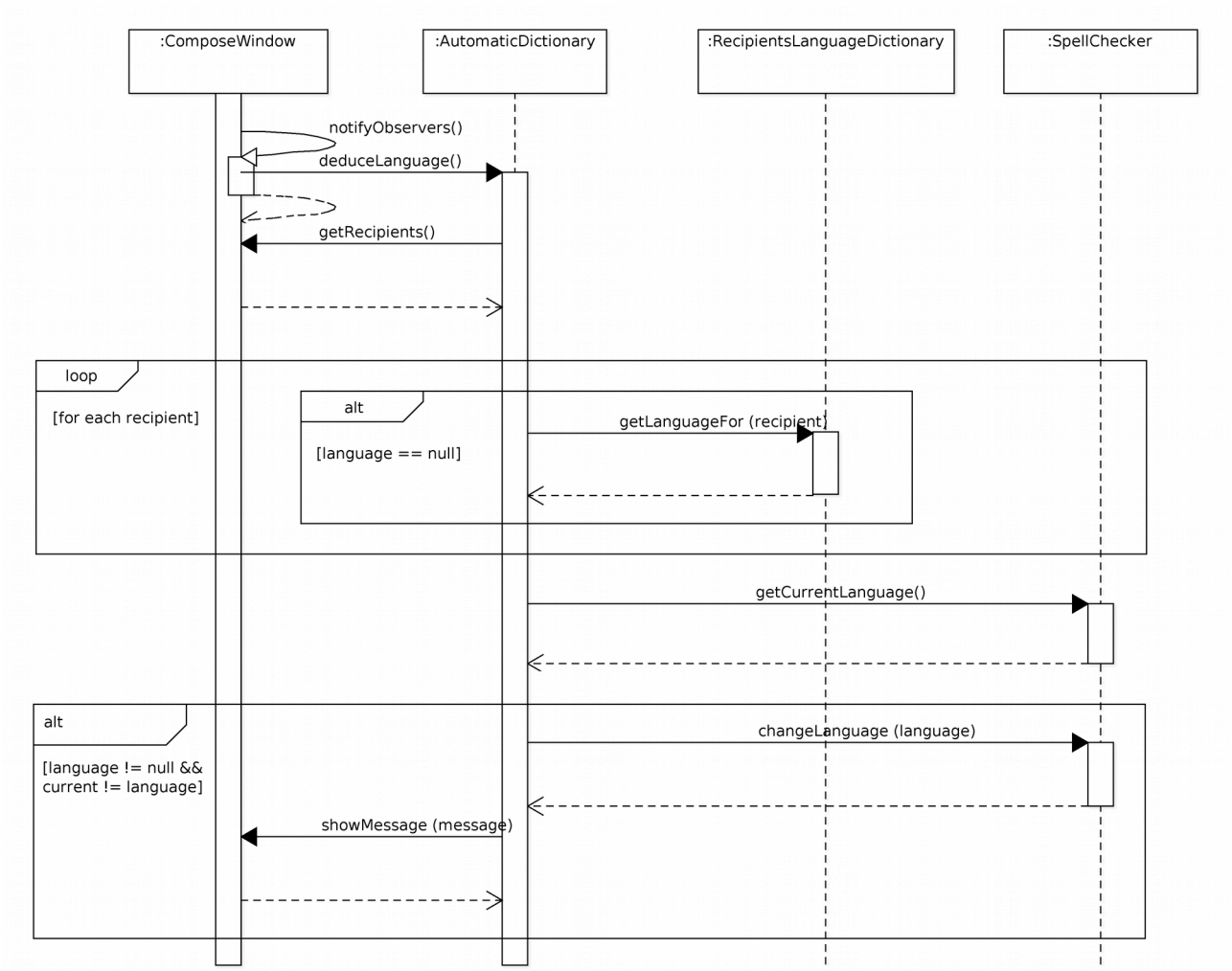


Illustration 20: recipients changed event sequence diagram

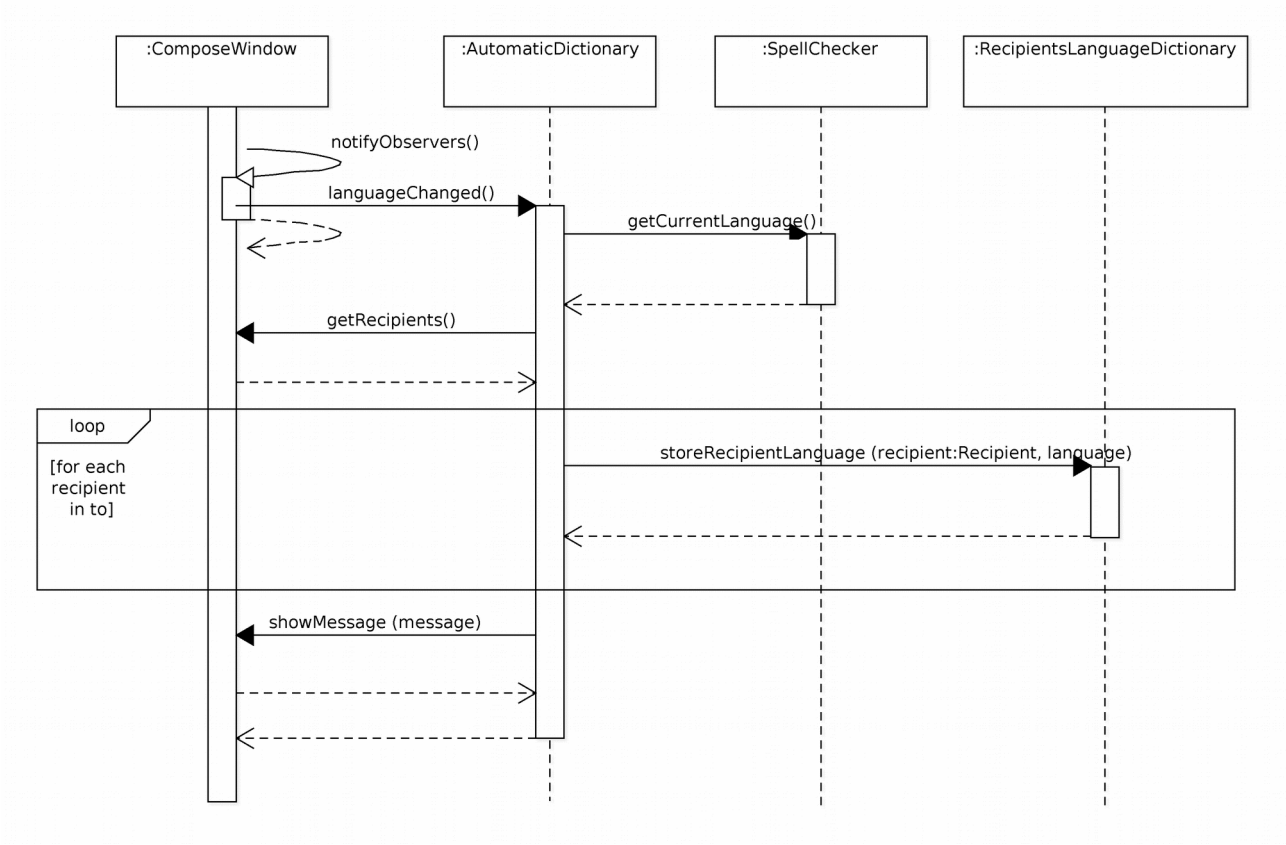


Illustration 21: language changed event sequence diagram

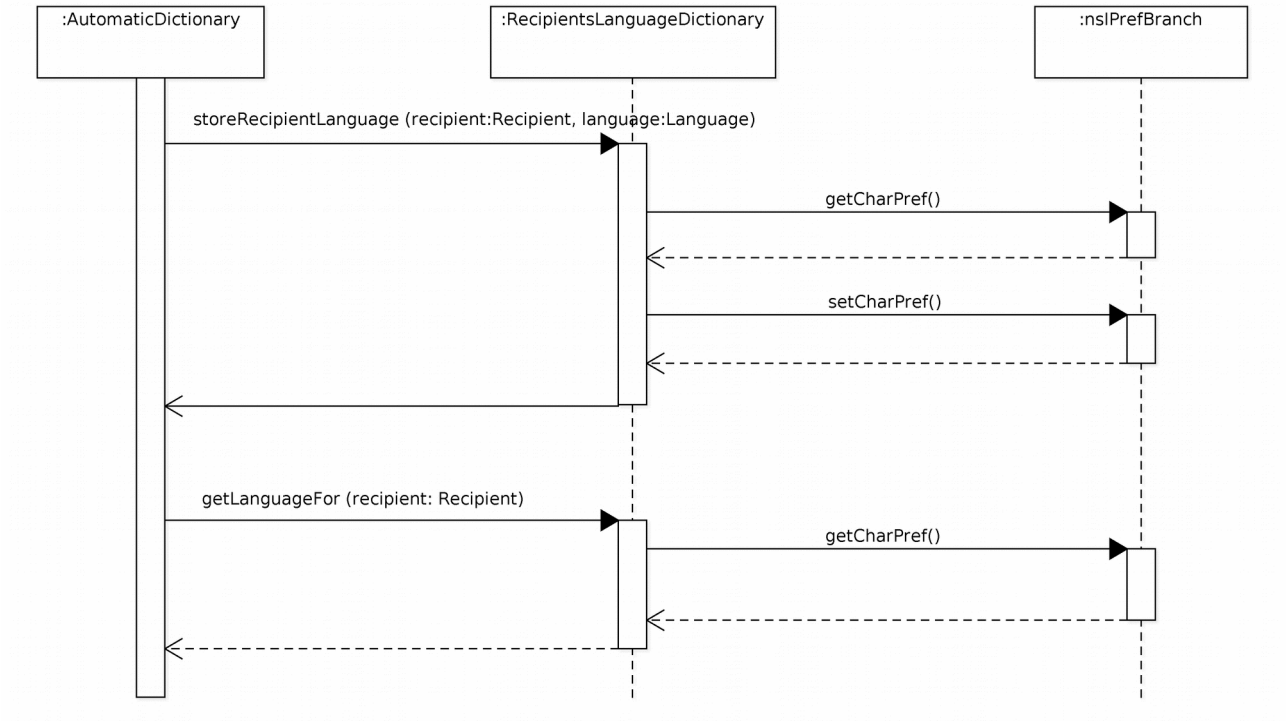


Illustration 22: RecipientsLanguageDictionary sequence diagram

5.4.1. Iteration 3

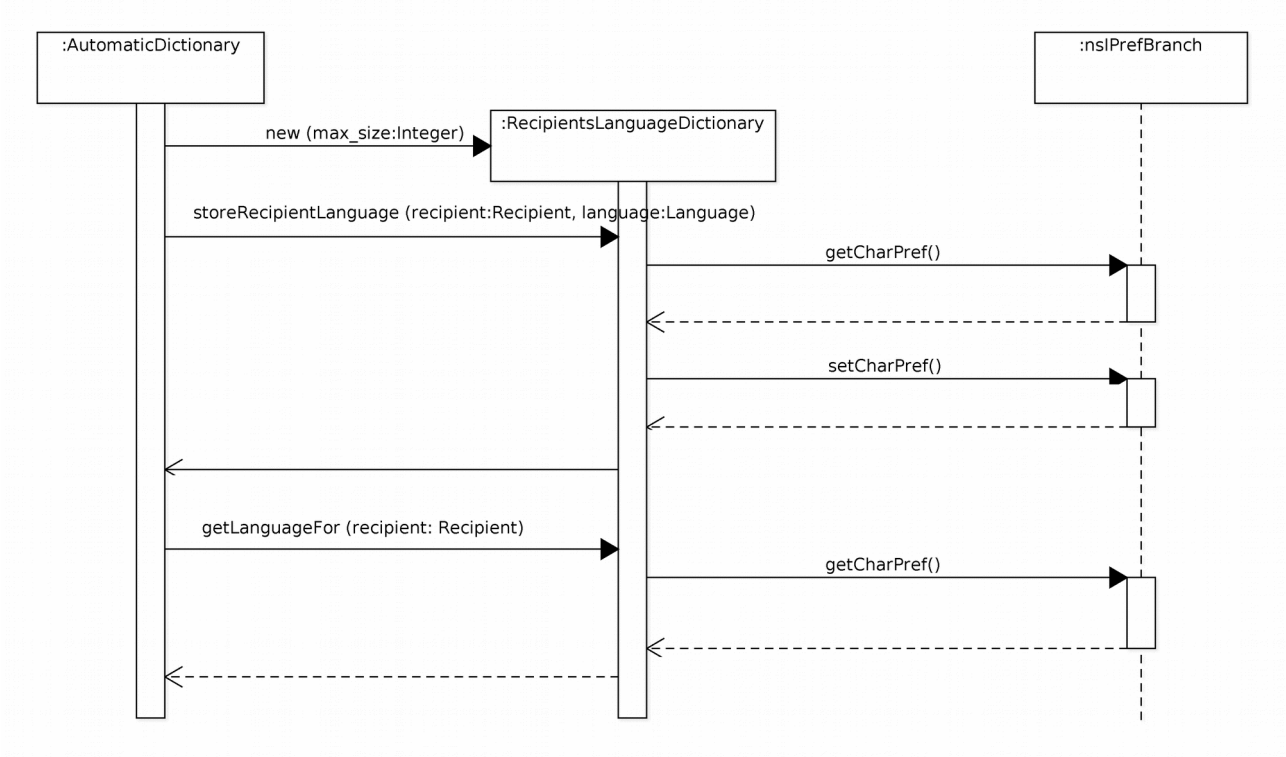


Illustration 23: RecipientsLanguageDictionary sequence diagram on iteration 3

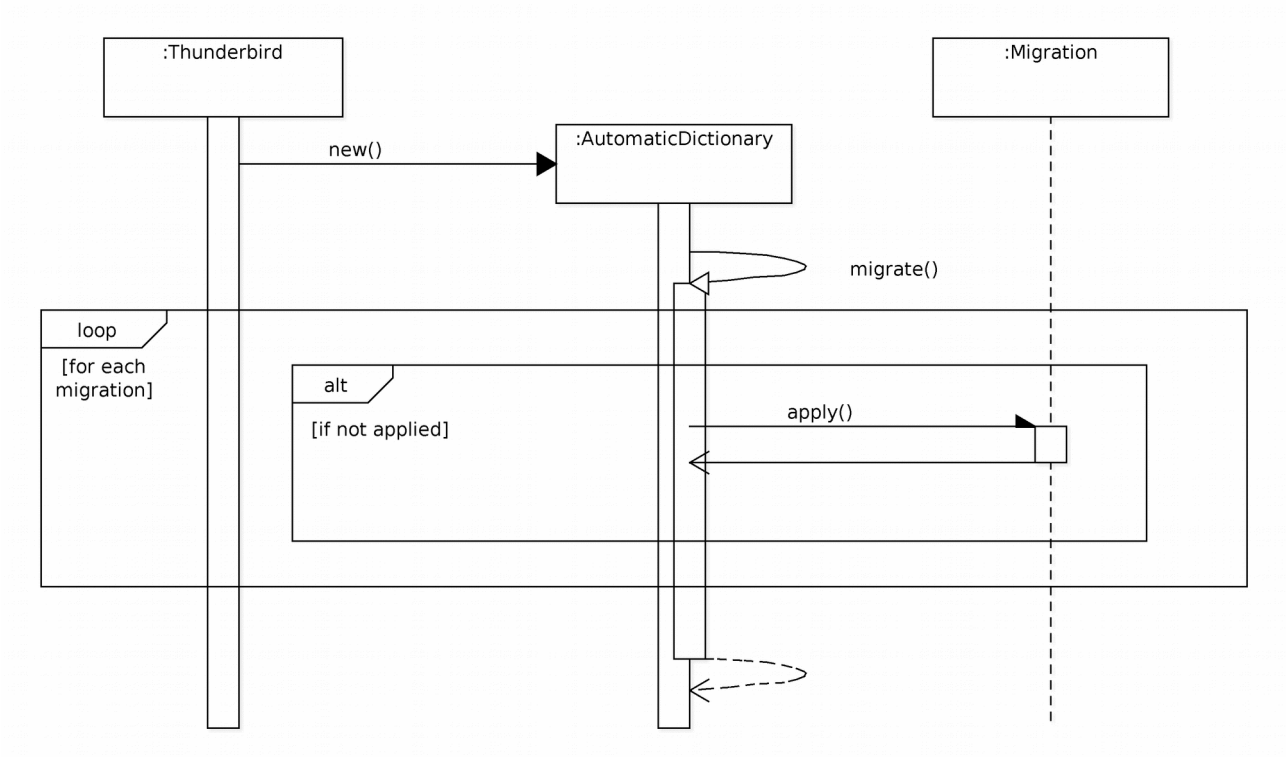


Illustration 24: AutomaticDictionary initialization sequence diagram

5.4.2. Iteration 4

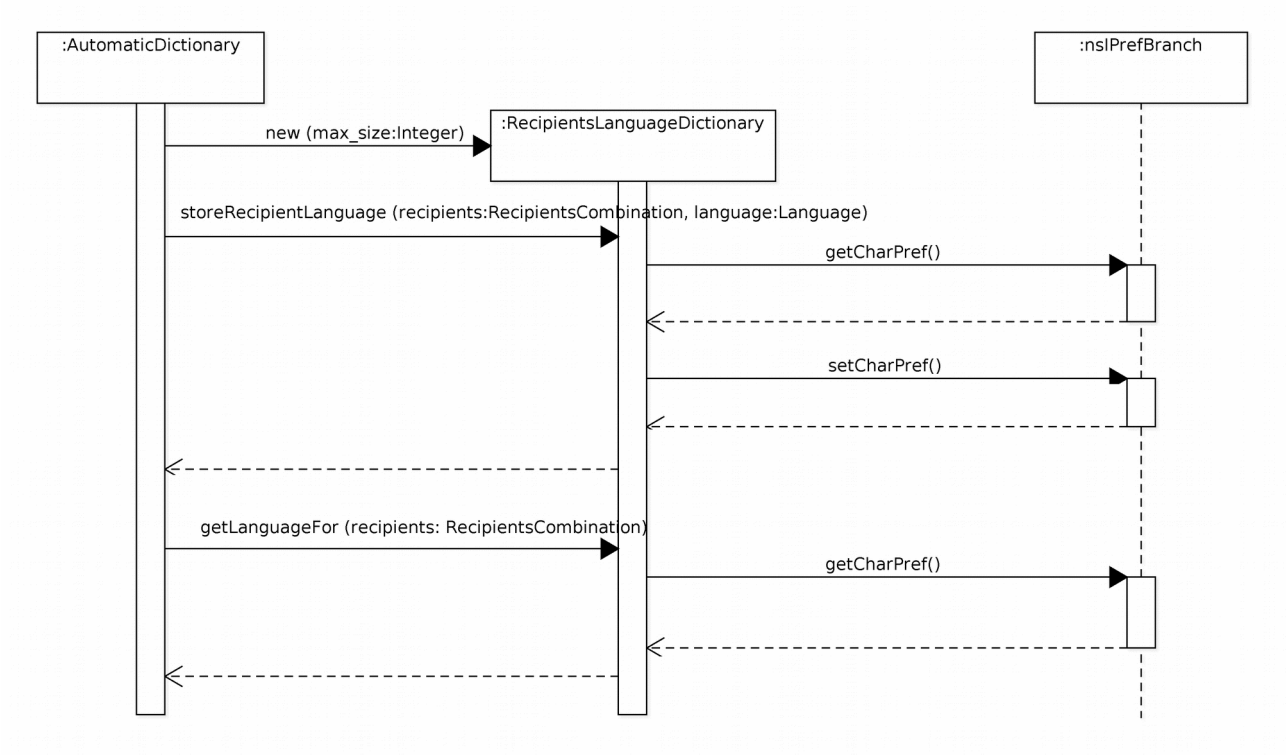


Illustration 25: RecipientsLanguageDictionary sequence diagram on iteration 4

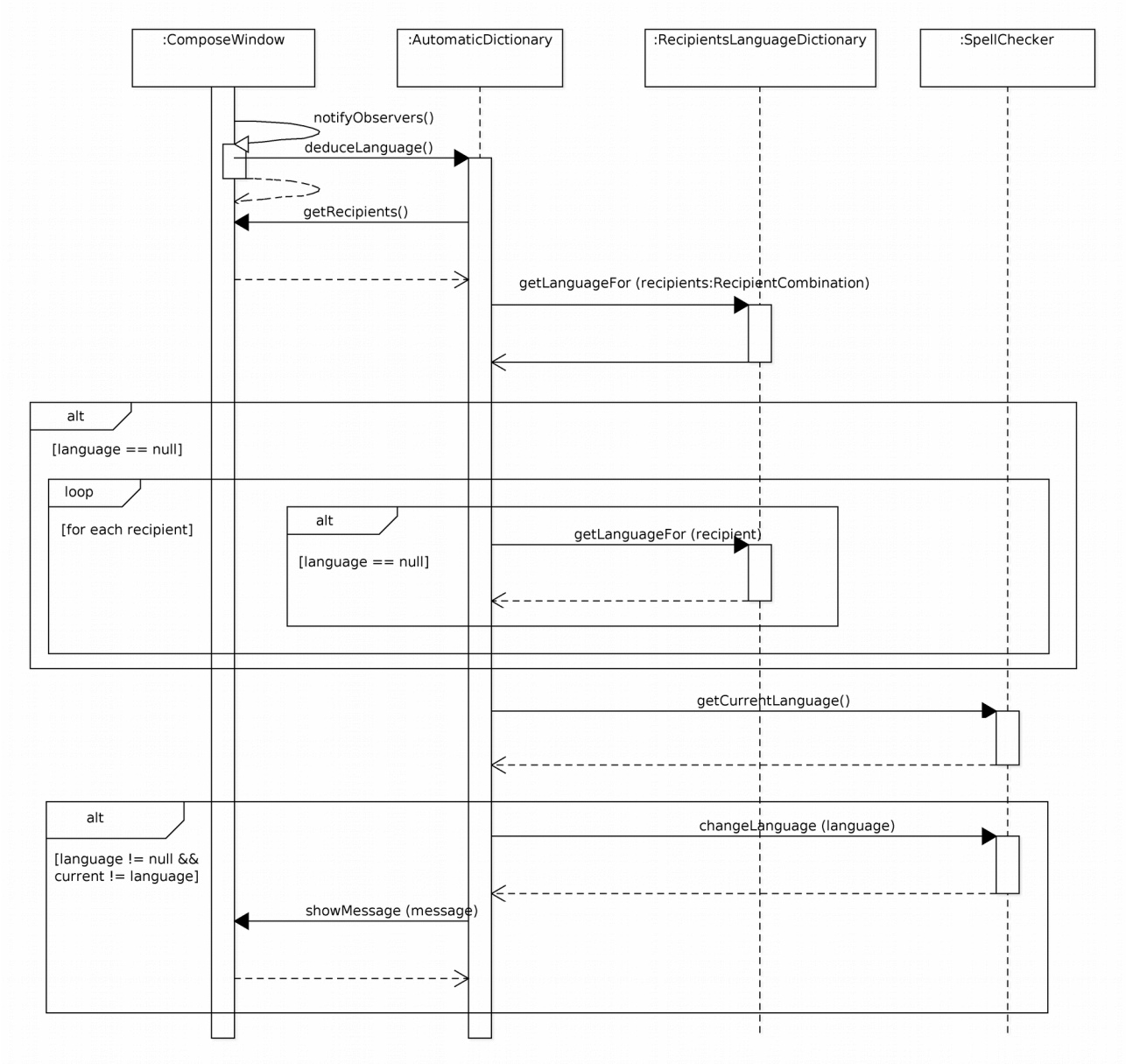


Illustration 26: recipients changed event sequence diagram on iteration 4

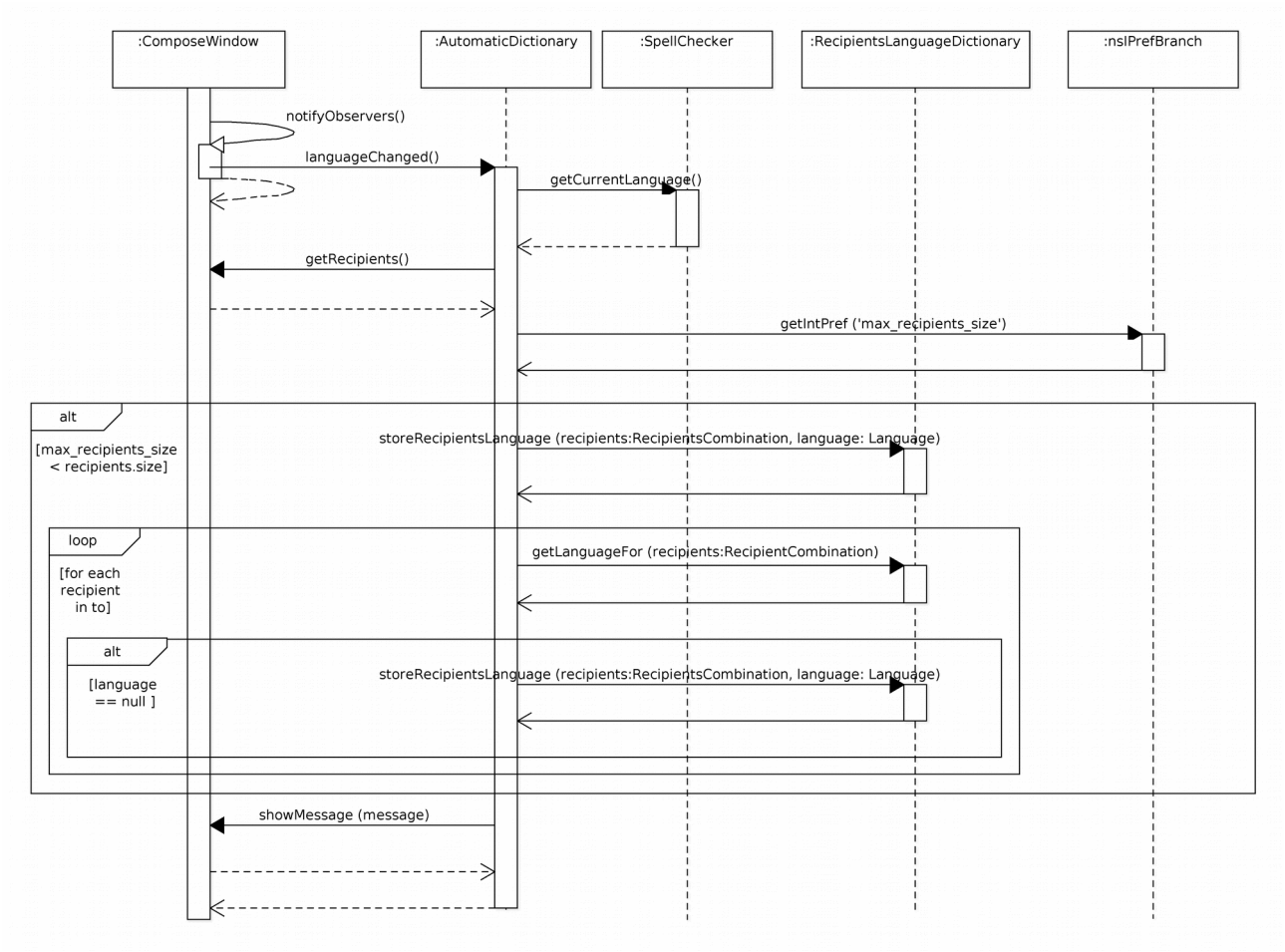


Illustration 27: language changed event sequence diagram on iteration 4

5.4.3. Iteration 5

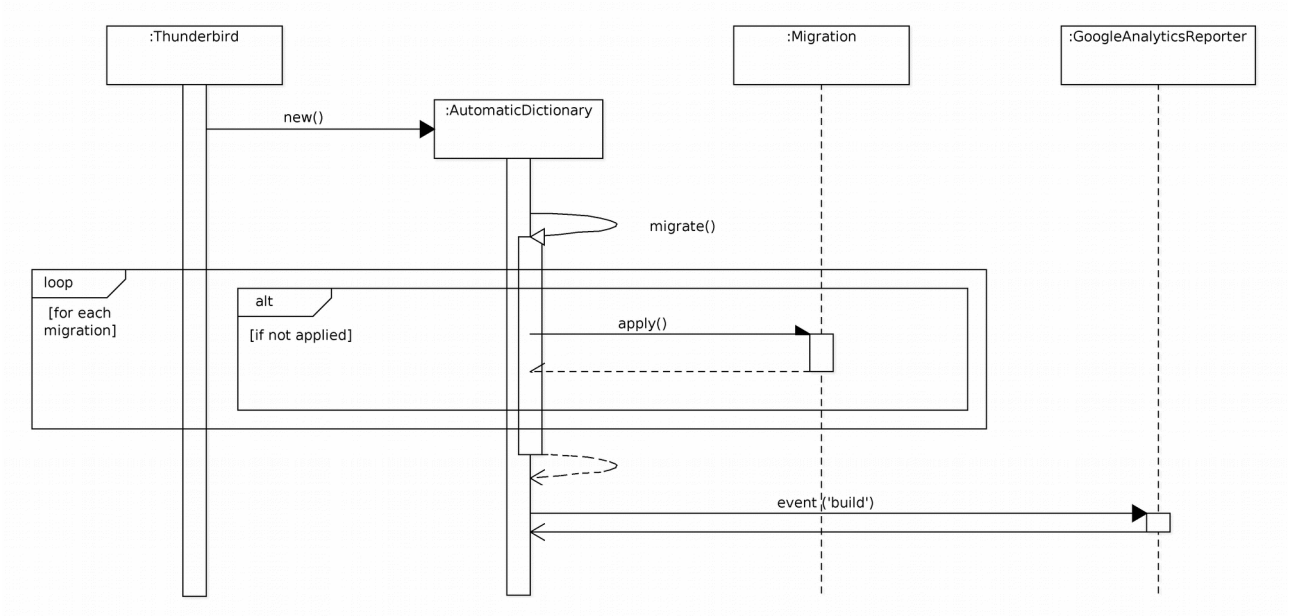


Illustration 28: AutomaticDictionary creation sequence diagram on iteration 5

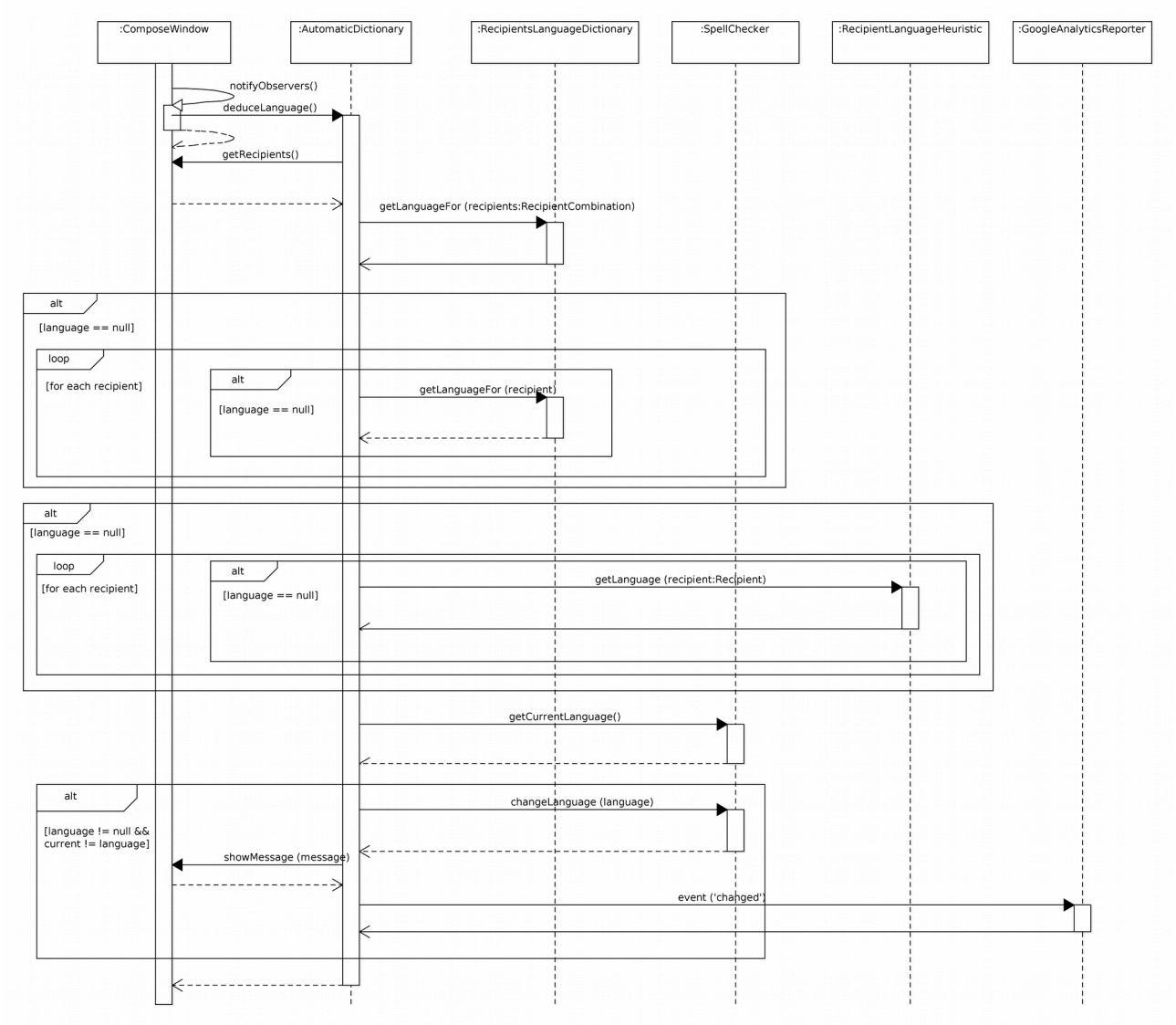


Illustration 29: Recipients changed event sequence diagram on iteration 5

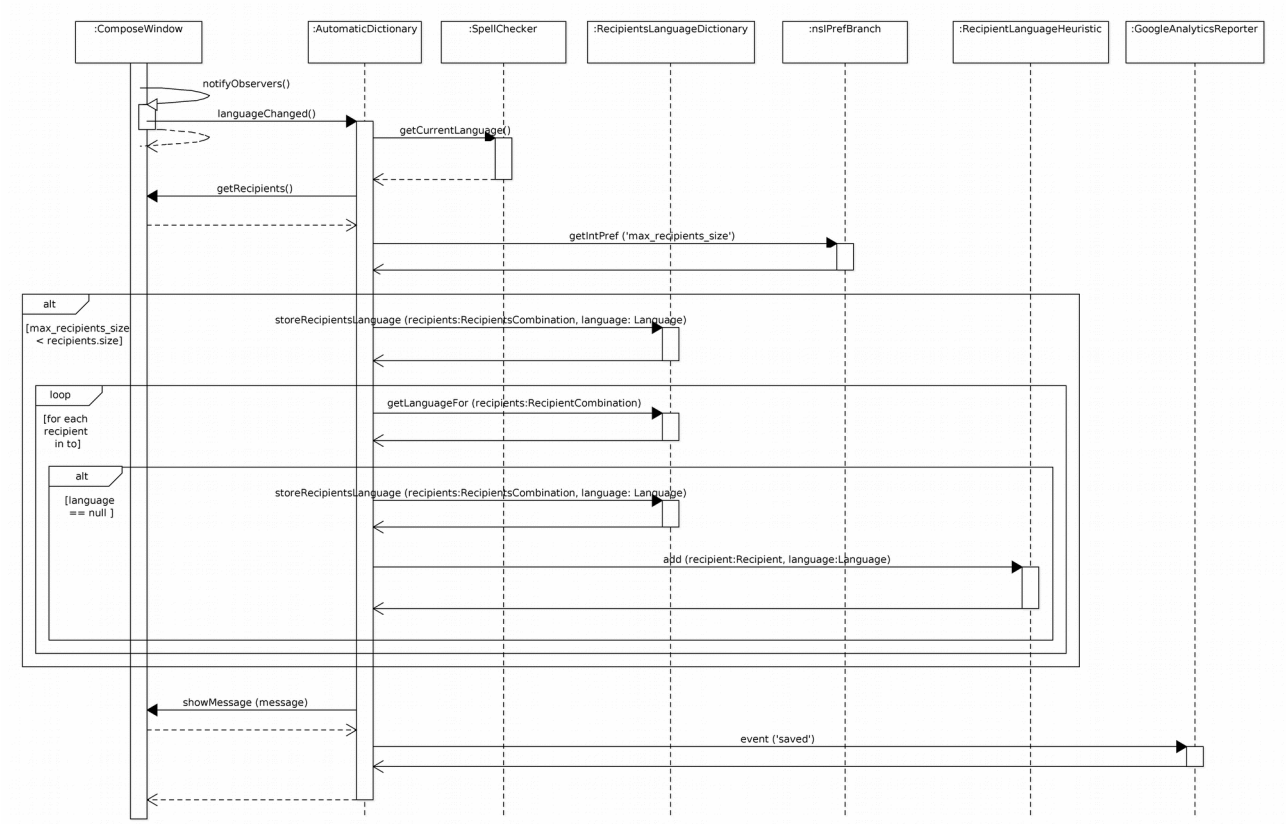


Illustration 30: Language changed event sequence diagram on iteration 5

5.4.4. Iteration 6

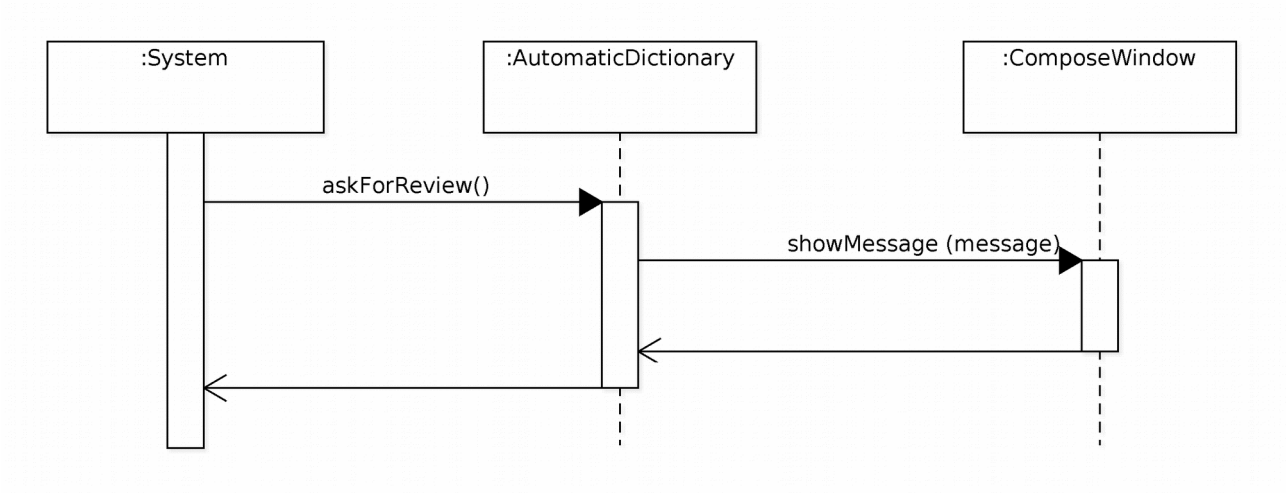


Illustration 31: Ask for reviews sequence diagram on iteration 6

5.4.5. Iteration 7

There is no sequence diagram change in this iteration.

6. Implementation

Once defined what we want to accomplish and the design of the iteration, we have to implement it.

The implementation had to follow the rules defined by Mozilla AMO Team [3], to keep the add-on ecosystem healthy. These rules are also thought to take care of the users.

Some of them, grouped by scope, are:

Technical:

- Provide reviewers with full source code
- Can't use unvetted third-party code libraries or frameworks
- Can't contain obvious coding errors
- Can't conflict with other well-behaved add-ons.
- Can't use APIs known to cause performance or stability problems

Security concerns:

- Can't cause harm to users data, systems, or on-line identities.
- Can't create or expose security vulnerabilities
- Can't tamper with the application/add-on update or block-list systems
- Can't execute remote code
- Can't degrade the security of HTTPS sites
- Can't install additional add-ons or system applications without user consent
- Can't include their own update mechanism

Privacy and User Consent

- Can't make unexpected changes to the browser or web content
- Can't prevent users from reverting changes made by the add-on
- Can't prevent the add-on from appearing in the Add-on Manager
- Can't prevent the user from disabling or uninstalling the add-on
- Can't send sensitive information to remote servers unprotected.
- Can't store browsing data from private browsing windows
- Can't leak identity information to web content in private browsing windows
- Can't change Firefox preferences without user consent
- Must clearly disclose all user data handling in a Privacy Policy

User Experience:

- Can't break or disable core application features
- Can't make any changes which persist after the add-on is disabled or uninstalled
- Has to be easy to use and provide a consistent user experience
- Has to appeal to a general audience
- Can't require payment to use core add-on features (upfront or after trial)

Content

- Can't violate the Mozilla acceptable use policy

6.1. Technology

6.1.1. View

The view in this Mozilla project is build based on XUL technology. [4]

If we needed to show complex data, we could render inside an HTML object or HTML Canvas tag, but it's not our case. Our communication with the user is very basic, so the best option is to use XUL.

XUL is also very easy to use if you have experience on HTML because it has a lot of similarities.

6.1.2. Programming language

Based on the restriction to work as a plug-in of Thunderbird, the possible choices for the programming language were:

- JavaScript
- C++

But they encourage developers to use JavaScript because it is portable to all supported platforms.

The only reasons that could lead us to use C++ are:

- High performance needs
- Use of third party libraries written in C or C++
- Use of Mozilla interfaces that are not exposed to JavaScript.

As we were not in any of those cases, JavaScript was chosen.

6.1.3. Storage

To persist data between user sessions and not share the stored data between user accounts, we have chosen the Preferences API [5].

This internal service has the following features:

– Key-value store of strings, booleans and integers.

- Keeps different user account data separated. It persists the data on user profile folder, where all emails and account private data is stored too.
- Decouple our project from platform specific file system access.

6.1.4. Localization

Thunderbird and Mozilla as a community are prepared to support different languages and localizations.

View with XUL

To localize view items, we can use entities that are defined in different files for each language.

We define a file with “dtd” extension with the content:

```
<!ENTITY menu_refresh_now.label "Refresh Now">
```

And in the XUL file we use it as an HTML entity:

```
<menuitem label="&menu_refresh_now.label;" oncommand="StockWatcher.refresh()"/>
```

In JavaScript

We create a “.properties” file that will define keys and values as strings.

```
changeString=Chg:
```

In our JavaScript code we load them using the StringBundle module:

```
this.string_bundle = new
StringBundle("chrome://automatic_dictionary/locale/strings.properties");
```

We have to use the folder with the locale name to separate each language.

```
chrome/locale/de/preferences.dtd
chrome/locale/de/strings.properties
chrome/locale/ca-ES/preferences.dtd
chrome/locale/ca-ES/strings.properties
chrome/locale/en-US/preferences.dtd
chrome/locale/en-US/strings.properties
chrome/locale/es-ES/preferences.dtd
chrome/locale/es-ES/strings.properties
```

6.1.5. Testing

To implement automated tests for the plug-in I wanted:

- Basic features like asserting equal, true, or false.
- Easy interface
- Mature source code
- Well documented

Extra features:

- Support integration testing. This means to start a Thunderbird and interact like a user would do and check the behavior.
- Be in JavaScript, to not increase the used languages in the project.

Back in 2009, after searching for a while I could not find a suite to do integration testing. After discarding this part of testing, I decided to implement the unitary test tool by myself.

I wrote a little library that defines “assertEqual” and other functions that did the trick. It was easy to implement and a good exercise to do.

As I wanted tests to run as close as Thunderbird JavaScript engine as possible, I tried SpiderMonkey [6], because it is what Thunderbird uses internally, but it did not have a feature I needed as a developer. Be able to see the stack trace of raised errors.

Based on that limitation, I did switch to “rhino” [7], a standalone JavaScript engine maintained by Mozilla that runs on the Java machine. It shows traces and shows where the error was generated.

6.1.5.1. Nowadays

I have to say that nowadays there are a lot of test suites in JavaScript, but back then in 2009, i did not find a suite mature enough or decoupled from a browser environment that could be used here.

If it were done now, maybe I'd choose Marionette [8] for integration testing.

And for the unit testing I would use mocha [9] because it has been widely used in the Node.js community.

6.2. Source code

All the source code is publicly available on GitHub and licensed under MIT License.

GitHub: https://github.com/beltrachi/automatic_dictionary

I used the MIT License because allows anyone to do almost anything with it. It's less restrictive for derivations or usages than others. There are others with this features but MIT is used a lot in open source communities.

The structure of the source is the one required for Mozilla XPIs (Cross-platform Installer Module).

This source code was reviewed manually by a member of the Mozilla AMO team so we can say that code is well written and follows their rules.

Here we have a sample code of the ComposeWindow class:

```
AutomaticDictionary.extend( AutomaticDictionary.ComposeWindow.prototype, {

    notificationbox_elem_id: "automatic_dictionary_notification",

    name: "ComposeWindow",
    logger: null,

    recipients: function( recipientType ){
        recipientType = recipientType || "to";
        var fields =
Components.classes["@mozilla.org/messengercompose/composefields;1"]
            .createInstance(Components.interfaces.nsIMsgCompFields);
        this.ad.window.Recipients2CompFields( fields );
        var nsIMsgRecipientArrayInstance = {
            length:0
        };
        var fields_content = fields[recipientType];
        if( fields_content ){
            nsIMsgRecipientArrayInstance =
fields.splitRecipients( fields_content, true, {} );
        }
        var arr = [];
        if(nsIMsgRecipientArrayInstance.length > 0){
            for(var i=0; i< nsIMsgRecipientArrayInstance.length; i++){
                arr.push(nsIMsgRecipientArrayInstance[i].toString());
            }
        }
        this.logger.debug("recipients found: " + arr.toSource());
        return arr;
    },

    getCurrentLang: function(){
        var spellChecker =
this.window.gSpellChecker.mInlineSpellChecker.spellChecker;
```

```
        var lang = spellChecker.GetCurrentDictionary();
        this.logger.info("gSpellChecker says current lang is "+lang);
        return lang;
    },

    showMessage: function( str, options ){
        options = options || {};
        var notification_value = "show-message";
        var nb =
this.ad.window.document.getElementById(this.notificationbox_elem_id);
        var n = nb.getNotificationWithValue(notification_value);
        if(n) {
            n.label = str;
        } else {
            var buttons = options.buttons || [];
            var priority = nb.PRIORITY_INFO_HIGH;
            n = nb.appendNotification(str, notification_value,
this.params.logo_url, priority, buttons);
        }
    },
```

Coding rules are defined by Mozilla in their web page.

A sample of the more basic would be:

- Functions are camel-case except for first letter which is lower case. Example: “getCurrentLang”.
- Classes are camel case. Example: “ComposeWindow”.
- Variables are underscored. Example: “fields_content”.
- Variables describe what they contain. No random names.
- Constants are in upper case. Example: “PREFERENCE_SCOPE”.

There are more rules on the Mozilla developer guide page [10].

I did not use any syntax linter like jslint [11] because I did not know any of them back on 2009. Now I would recommend it to reduce development time as it detects bugs without running the code at all.

6.3. Development environment

A development environment needs these features:

– Representative of real environment.

– Changes on code can be tested fast and easy.

To do so, we did:

1. Clone the extension repository.
2. Install Thunderbird.
3. Create a development account.
4. Go into the folder of the account and install a “fake” add-on. The steps are:
    1. Create a file in your extensions folder on your development account. Example:

        ```
        touch
        ~/.thunderbird/xa5lv6p8.devel/extensions/automatic_dicti
        onary_extension@jordi_beltran.net
        ```

    2. Edit the file and set as content the path to your cloned repository. In my case: '/home/jbeltran/workspace/automatic_dictionary'
5. Start Thunderbird in command line in case any message appears.
6. Open the Thunderbird “Error Console” to see any messages.

With this setup we are avoiding to build the package and install it on the account every time.

To test a change now you only need to stop and start Thunderbird and the changes will take effect.

Debugging

To debug a context or scenario, you can add loggers and read the output on the Error Console.

Inspecting the view

To inspect the interface of Thunderbird and see the XUL nodes, there is a very useful add-on named “DOM Inspector”.

With this add-on we could see which XUL tag was holding the recipients and find a way to read them.

6.4. Release process

To deploy a new release to the users, we need AMO Team to publish the new version on the site and distribute it on all users. The flow would be this:

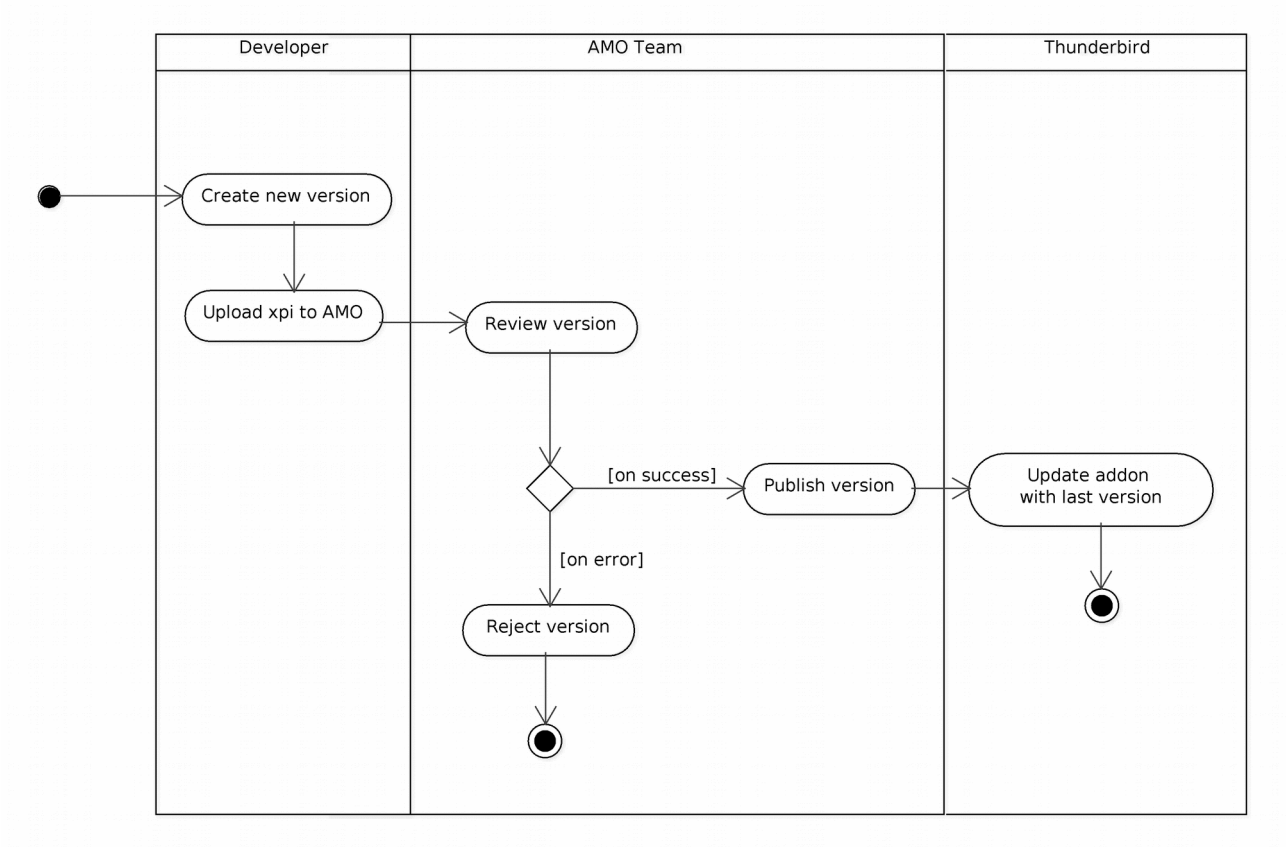


Illustration 32: Publish new version sequence diagram

6.5. Data structures

For the implementation of the data structures, we defined what requirements we wanted and designed some tests to check the behavior and that it was performing well.

6.5.1. Recipients Language Dictionary

The main structure we needed was a dictionary where we could use RecipientCombination as keys and Language as values.

This structure also needed to know the least recently used data to expire this and not the others.

To implement this we defined first the interface and some tests.

To simplify this interface, we defined a serializer function that converted RecipientCombination into string, outside of this structure.

We also needed methods to convert that structure into string to have persistence and read it back when user starts Thunderbird again.

Interface:

```
# Create a new structure with that max size
new(Integer max_size): RecipientCombinationDictionary

# Retrieves a language for that recipient combination
get(recipientsString String): Language

# Sets or updates a language for that recipient combination
# This method also can expire the least recently used pair
# when it reaches max size
set(recipientsString String, Language l)

# Returns the number of pairs stored.
size(): Integer

# Serialize structure to string
toJSON(): String

# Create structure from string
fromJSON(String json): RecipientCombinationDictionary
```

Based on this interface we built a first version that implemented the structure with a JavaScript hash and an array to keep the keys sorted. Like this:

```
hash = {};
order = [];
key = “foo@bar”
language = “ca”
hash[key] = language
order.push(key);
```

But this structure had operations with big cost (O(n)), mainly when we removed an item or updated the order array, we needed to recreate the full order array without the removed item and that was expensive (O(n)).

So we needed to fix that aspect.

The next approach was to use a sorted set to store the order. Also this set would keep insertion order and have fast access by key. JavaScript did not have a LinkedHashSet structure like Java does so we had to implement it and use it in our dictionary structure.

The interface of the sorted set is:

```
push(String key, String value)
remove(String key)
contains(String key): Boolean
first(): String
last(): String
toArray(): Array
```

The implementation of this structure, like the others was done in test driven development. First I wrote tests that checked basic structure operations and then wrote the code that fitted those tests.

Once we had that structure we could create the second generation of the dictionary.

```
hash = {};
order = new LinkedHashSet();
key = “foo@bar”
language = “ca”
hash[key] = language
order.push(key);
```

This time that structure passed the performance tests that made sure that loading and dumping the structure with 1000 elements was loaded and unloaded in less than 500 ms.

The plug-in was designed to work with 1200 pairs at most so this was inside reasonable margins.

6.5.2. Recipient Language Heuristic

This structure had some requirements that made it more complicated than expected. We wanted to store an Internet domain relation to a language. As a domain can have more than one language used we designed a counter for each language. The resulting structure is, for a domain, count how many occurrences of each language were found.

An example:

```
alice@example.com => “en”
bob@example.com => “en”
john@example.com => “es”

The data stored in our structure would be:
“example.com” => { “es”: 1, “en”: 2 }
```

Also, we wanted the plug-in to detect country domain languages automatically. It would learn by itself that “.it” domains were for “Italian”, and the same for “.es” and “.cat”.

To do that, the behavior had to be as follows:

```
alice@example.es => “es”
bob@example.it => “it”

So data would be:
“example.es” => {“es”: 1}
“example.it” => {“it”: 1}
“.it” => {“it”: 1}
“.es” => {“es”: 1}
```

When we ask for a language for “other.es” it will return “es”, even if it does not know “other.es” because it has counters for domain suffixes.

The interface would be as follow:

```
add(String key, String value)
remove(String key, String value)
get(String key)
toJSON(): String
fromJSON(String data): RecipientLanguageHeuristic
```

To implement this we used a tree structure in JavaScript where each node stored a counter for each language.

If we represent it in a plain hash, it would be as follows:

```
{
    "_counters" {
        "es": 2,
        "en": 1
    }

    "es": {
        "_counters": {
            "es": 2
        },
        "example": {
            "_counters": {
                "es": 2
            },
        },
    }
    "com": {
        "_counters": {
            "en": 1
        }
        "example": {
            "_counters": {
                "en": 1
            }
        }
    }
}
```

The real implementation was made using a tree where each node had logic to process the add and remove locally.

The structure was build successful and the performance tests proved that was fast enough to release it to the users.

7. Verification

The extension is being used by 3.000 unique users per day and has been used since 2010. This is the best verification we could have. And also the reviews received from the users.

Besides that, to verify each iteration we had two set of automated tests, unit testing and functionals. We also had user feedback that detected cases that we did not detect.

Unit testing

This level of testing verifies that each piece of the system operates correctly and as expected but isolated from the rest.

We create tests that check each scenario the unit is expected to support and check the response.

This has been done with a little library that makes easy to check that a call or expression matches what is expected.

In each iteration we added the tests to cover the code added.

Functional testing

There was a part of functional testing that was covered by automated tests in JavaScript, but we needed to test manually the integration of the add-on with Thunderbird because we did not know the way to automate it.

Community feedback

In this project, we had a third verification data source, the user feedback.

This data came from user reviews, issues opened on GitHub, and users that contacted me by email.

This gave me a very valuable feedback and let me fix bugs that I'd haven't detected by myself. This includes:

- Slower platforms

- Interaction with other plug-ins like Conversations or Dictionary Switcher.
- Add-on preferences setup that we did not expect users to configure that way.
- Thunderbird configurations like disabling the spell-checker.

7.1. Iteration 1

In this iteration we wanted to implement the most basic features and learn how to integrate the extension with Thunderbird.

The requisites of this iteration were:

- Integrate with Thunderbird application
- Be able to save and restore a language for a recipient.

Issues with this version:

1. Does not work well when more than one compose window is open at the same time.
2. Thunderbird reuses objects and windows, and that was messing with the add-on initialization. First time you opened a compose window it worked well, but second time it did not work as expected.

Conclusions:

1. Iteration completed successfully. When we add the plug-in to Thunderbird, it fits the expected behavior.
2. The issues raised should be fixed on next iteration.

7.2. Iteration 2

In this iteration we focused on fixing the bugs and publishing the add-on to the community.

The requisites of this iteration were:

- Fix previous bugs.
- Make add-on translatable (Localization non-functional requisite)
- Get AMO certification to publish the extension to all Thunderbird users. (Certification non-functional requisite)

7.2.1. Testing results

1. Known Issues were fixed
2. Add-on is translatable.

7.2.2. AMO Certification results

This version received a “preliminary review”. This is a lighter review and involves the editor reviewing the source code for malicious code and security problems, but without checking for 100% policy compliance or testing the add-on thoroughly. Add-ons that undergo preliminary review will have cautions placed on their install buttons and have a slight penalty in search results.

The result from that review was **successful**.

The approval came with two technical notes:

1. *“If you want to share non-persistent stuff between windows the best method would be to use a JS module.”*

   This hint is right, although we did not share non-persistent stuff between windows. Each window has it's own plug-in instance by design and they share data each other through the data layer.

2. *“The whole locking is unnecessary. You don't have a parallel access problem as all (chrome) code of all windows runs on a single thread. Unless you "reschedule" (e.g. setTimeout/timers/postMessage) in the middle of your "commits" you will be safe without locking.”*

   In fact we do setTimeout/timers, so I left the lock logic there.

This confirms that the reviewer did take the task seriously and reviewed the code thoroughly.

7.2.3. Community feedback

The user response can be considered very positive. We can see that by the daily users graph.

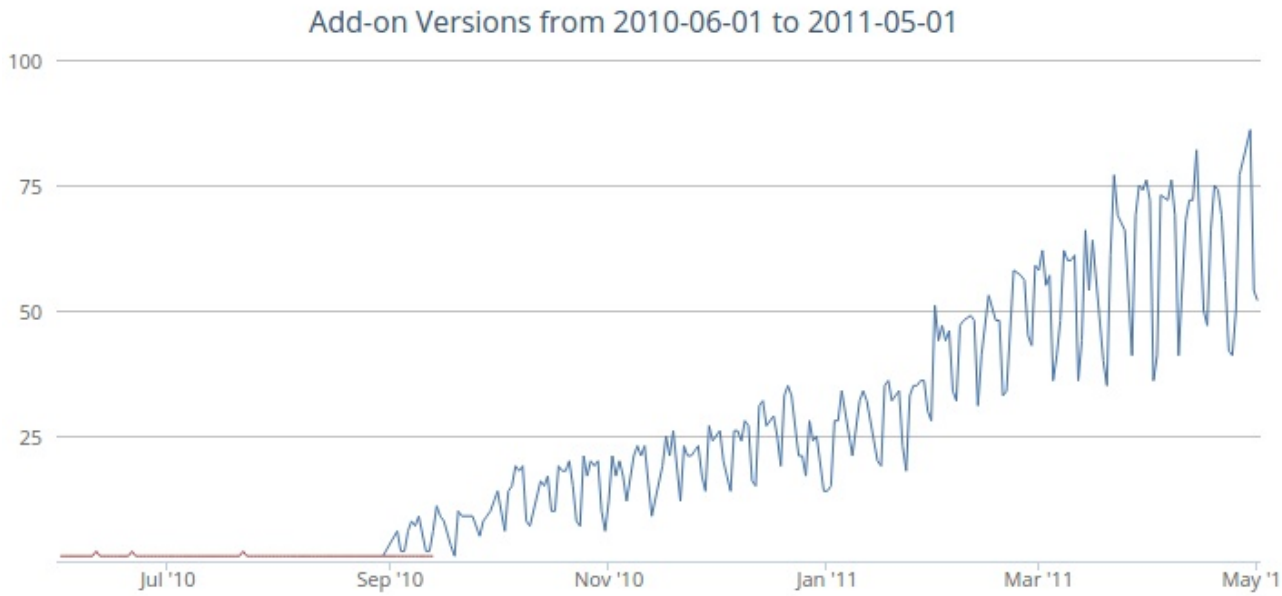


Illustration 33: v1.0.1 daily users evolution

7.2.4. Conclusions

1. Iteration completed successfully.
2. The add-on is having good acceptance, at least people is not removing the add-on after installing it and they are not leaving bad reviews.
3. We can go on with our road-map.

7.3. Iteration 3

The requisites of this iteration were:

- Internationalization
- Speed and efficiency
- Upgradeability

7.3.1. Testing results

1. All functional tests were passed
2. Add-on data was migrated successfully from an old installation (Upgradeability)
3. When data reaches max_size of 200, it removes the least recently used item from the recipient-language pairs dictionary. (Speed and efficiency)

We cannot confirm that adding Spanish and Catalan localization affected on installations because Mozilla did not collect statistics on that back on 2011, but we believe that having an add-on in your language is something users like.

7.3.2. AMO Certification results

In this iteration we requested a full review of the plug-in. This review is more complete than the last one and includes real functional testing from a member of AMO Team.

The review was successful and this time we were listed on the add-ons site with no warning message. This is expected to boost user installation.

7.3.3. Community feedback

After this release, users migrated to 1.1.0 gradually with no reported incidents.

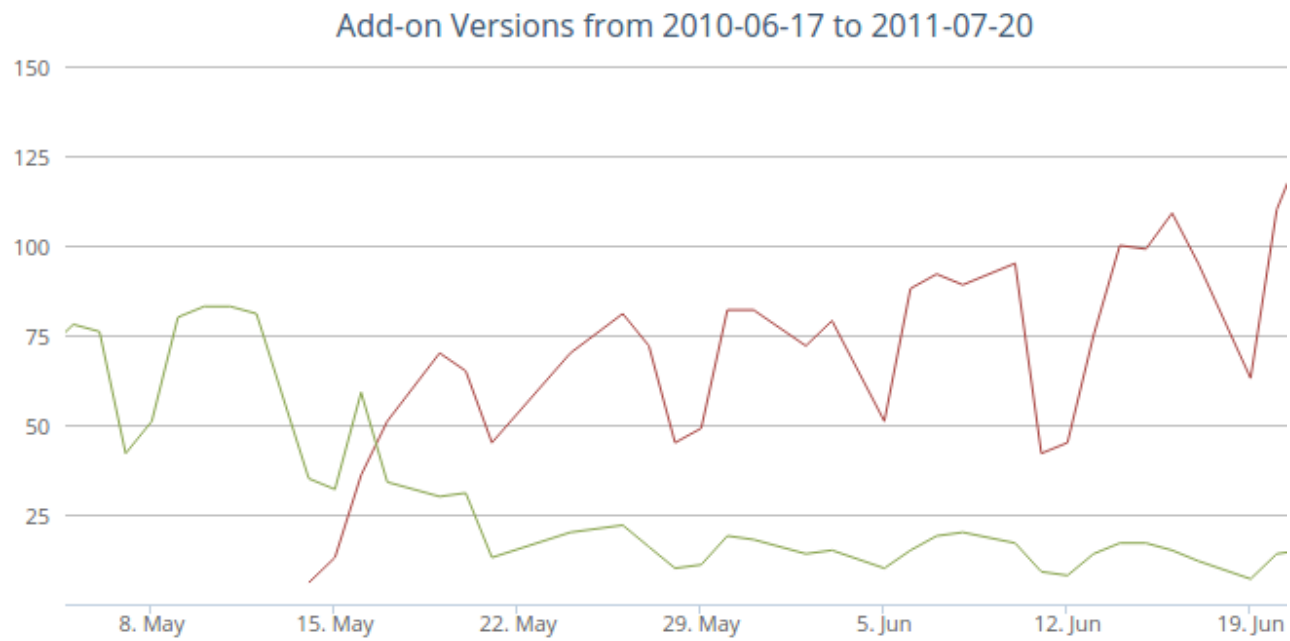


Illustration 34: Active users migration to v1.1.0 release

7.3.4. Conclusions

1. Iteration completed successfully.
2. Publishing the add-on with full review has had no big effects on active users by now.
3. We can't verify localization effort effects. We'd need detailed statistical data on this. (Mozilla is offering it now on 2016 but not on 2010).
4. We can go on with our road map.

7.4. Iteration 4

The targets of this iteration were:

- Make add-on smarter. Work better on certain cases.
- Have a preference setup editable by user.
- Avoid polluting the storage with useless recipients.

7.4.1. Testing results

1. All functional tests passed.

2. Plug-in has a preferences window where we can edit preferences values.

7.4.2. AMO Certification results

This version passed certification successfully.

7.4.3. Community feedback

Users updated to this new version without incidents but users reported some bugs we describe later.

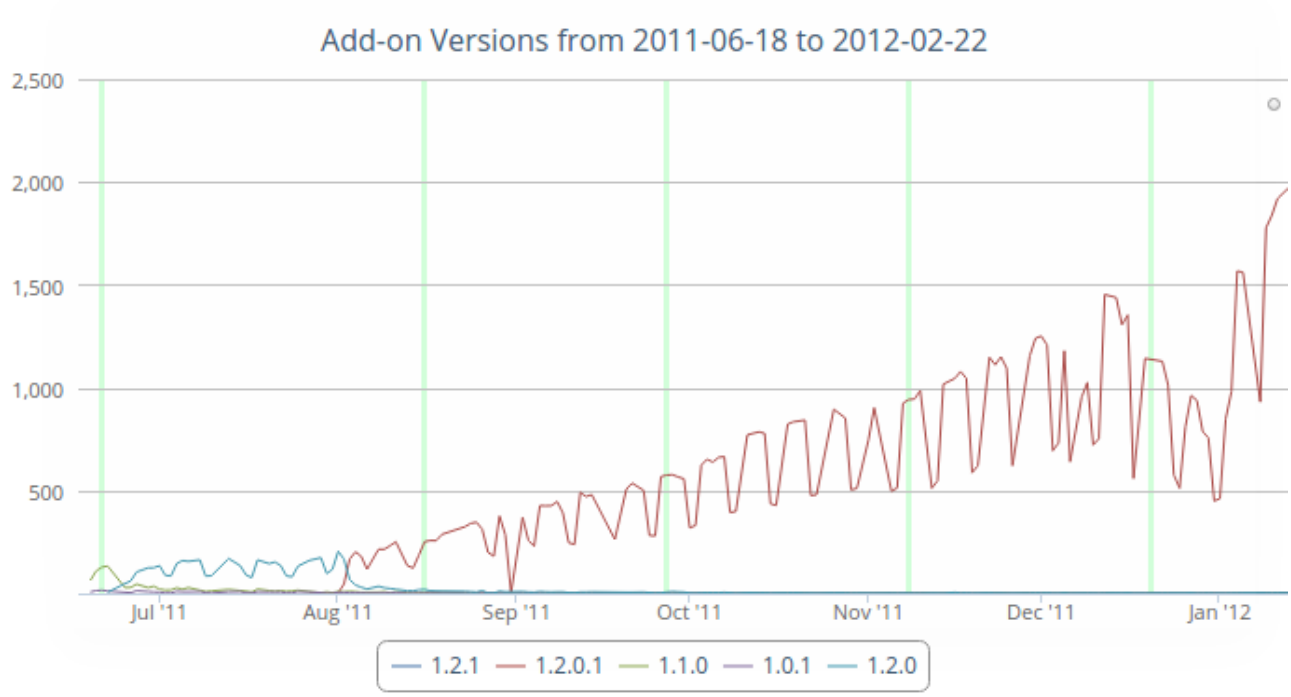


Illustration 35: User response to the release of v1.2.0

7.4.3.1. Reviews

The first review of the project was this:

> *Nice Idea, never worked*
> Rated ★★
> on November 19, 2011
>
> *Although the idea is great for everyone who has to write in different languages regularly, I've never got this add-on to work. When using this with the add-on dictionary switcher or simply selecting a different language via options->spell check->select language, it would always tell me that the language has not been saved.*

This negative review came because the user was using this add-on together with another that also modifies the spell-checker language.

It's important to give users honest feedback. This was my reply:

> Not tested with dictionary switcher
> by beltrachi (Developer) on November 21, 2011
>
> *I'm sorry buy I hadn't tested with dictionary switcher yet, as I didn't use it. After your review I checked it and saw that my add-on does not detect when the language is switched by the dictionary switcher.*
>
> *My add-on works if you choose the language, on the upper "Spellchecker" button selector, not in options > Spell check. Maybe I've not insisted in this enough. I've updated the plug-in description.*
>
> *Thanks for your time.*

Another review on this period was this:

> Simple but convenient
> Rated ★★★★★
> on November 21, 2011
>
> *I used to switch between English and Dutch spell checking many times per day. First I was looking for an add-on that would recognize the language I was typing, but this perhaps even works better.*

7.4.3.2. Thunderbird Releases

Between this release and the next, there were 2 Thunderbird releases that required me to release a new patch version.

The first was Thunderbird AMO team that notified me about the need to update my plug-in install.rdf. Releasing v1.2.0.1 on January 23th 2011.

The second time, a user opened an issue on GitHub reporting an incompatibility with the new Thunderbird 9. After reproducing the bug, I released a fixing version v1.2.1 on March 8th 2011.

7.4.3.3. Users contacted by email

Two composing windows bug

We received an email of a user explaining a bug. It was that the add-on did not work well when two composing windows were opened at the same time with different languages.

The plug-in did not take into account that maybe in another window the language could have changed.

The release 1.2.2 fixed this issue and the user left me a 5 star review as a reward.

Rated 
on March 10, 2012

I had a very nice conversation with the developer Jordi and helped him to improve the plug-in. In the most recent version (1.2.2), the add-on works just perfect. Thank you for this helpful add-on :-)

Request to implement heuristics

Another user, contacted me requesting it to be more clever and guess the language for a recipient based on the other emails from the same domain.

I did ask him to write an issue on GitHub with his request and so he did.

7.4.4. Conclusions

- Iteration was completed successfully.
- Users are starting to give feedback and is very useful and they are very kind.
- Plug-in has a bigger user base day by day.
- Thunderbird releases can break our functionality. We must assume that.
- Maybe we should give more priority to heuristics because users are asking for it.

7.5. Iteration 5

The objectives of this iteration were:

- Collect usage data
- Implement an heuristic to guess on unknown recipients.

7.5.1. Testing results

1. All functional tests passed.
2. Usage data collection can be disabled from preferences window.

7.5.2. Beta testers

As a user contacted me asking for heuristics, I asked him if he could test a beta release with that feature. The code was the exact one I was going to release, but we shared it with him before, in case he detected anything unusual.

We released it after he confirmed that it was good.

7.5.3. AMO Certification results

This version passed certification successfully.

7.5.4. Community feedback

Users updated successfully and Google Analytics started to show activity. And grateful reviews keep coming.

Rated ★★★★★
on February 16, 2013

I was waiting years for this! Awesome.

Rated ★★★★★
on March 8, 2013

Great!

7.5.4.1. GitHub issues

Compatibility with "conversations"

A user requested the plug-in to be compatible with another extension that shows a mail chain all together in a same page.

It's a request we should consider because this plug-in has more than 50k unique daily users.

7.5.5. Anonymous usage data

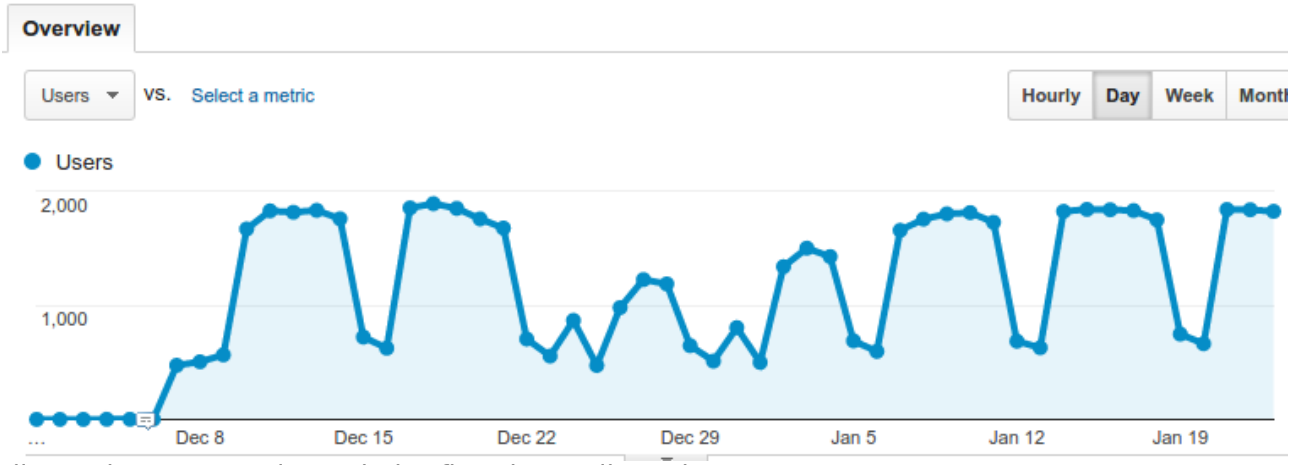


Illustration 36: Google Analytics first data collected

	Language	Sessions	% Sessions
1.	es-ES	107,314	26.77%
2.	en-US	82,990	20.71%
3.	de-DE	38,487	9.60%
4.	en-GB	28,337	7.07%
5.	fr	27,306	6.81%
6.	it-IT	26,085	6.51%
7.	pt-BR	12,907	3.22%
8.	es-AR	12,414	3.10%
9.	pl	8,568	2.14%
10.	ca	7,821	1.95%

Illustration 37: Google Analytics language report

Thanks to this data, now we can decide better which language we should add next or other decisions that could be influenced by this data.

7.5.6. Conclusions

- Iteration was completed successfully.
- Statistical data gives us important information to decide next actions.
- Giving users what they want can be better than giving them what you think they need. In this case we got a 5 star review for completing that feature.

7.6. Iteration 6

The objectives of this iteration were:

- Ask users for reviews.
- Compatibility with Conversations add-on.
- Restart-less.

7.6.1. Testing results

1. All functional tests passed.
2. Conversations can be used with this extension
3. No need to restart Thunderbird after installing the plug-in to use it.

7.6.2. AMO Certification results

This release did not pass the certification. The reviewer found an issue when opening the preference window and requested a fix.

Comments:
This version didn't pass full review because of the following issues:

1. After installing Error Console prints:

Error: TypeError: target is null
Source File: chrome://automatic_dictionary/content/lib/shutdownable.js
Line: 7

You need to correct them to get full approval. Thanks.

We prepared a new version that would fix this but this time another reviewer rejected it with this:

> This version didn't pass full review because of the following issues:
>
> 1) This seems not to work for me. Whenever I enter an address, I get an infobar saying "No language saved for these recipients". Changing the language does not trigger an infobar saying the language has been saved, as described, and composing a new message to the same author results in the aforementioned "No language saved" message.
>
> 2) Your metrics infobar should have an option to disable sending metrics.
>
> You need to correct them to get full approval. Thanks.

The reason of the point 1 was that she was using the “right click” on the text to choose another language, and that was something the extension did not support at that time. I prefered to add support for this instead of explaining to the reviewer how it worked because I believe that a lot of users could have the same misunderstanding.

The point 2 could be a little subjective, because there is no place on the Mozilla rules where they require this, but as Mozilla asks that a plug-in has to be easy to use and we should not do anything the user do not expect, I decided to implement the option on the metrics warning to disable it easily.

I released a new version (v1.5.0) with these changes and it was finally approved.

7.6.3. Community feedback

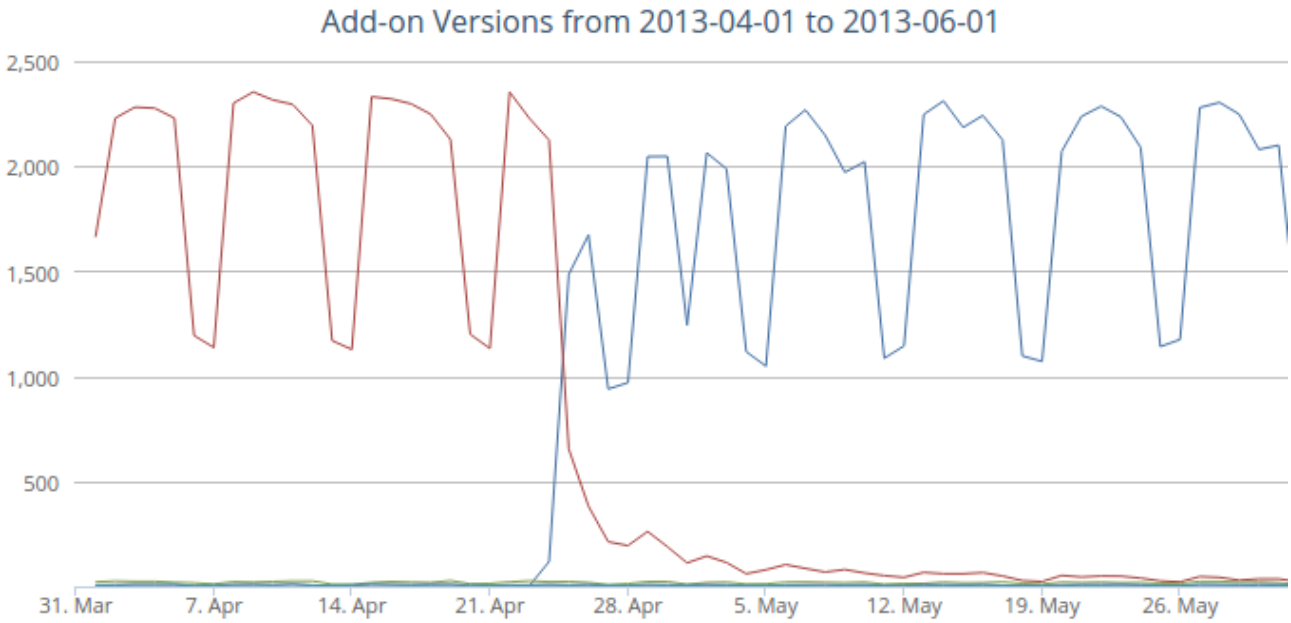
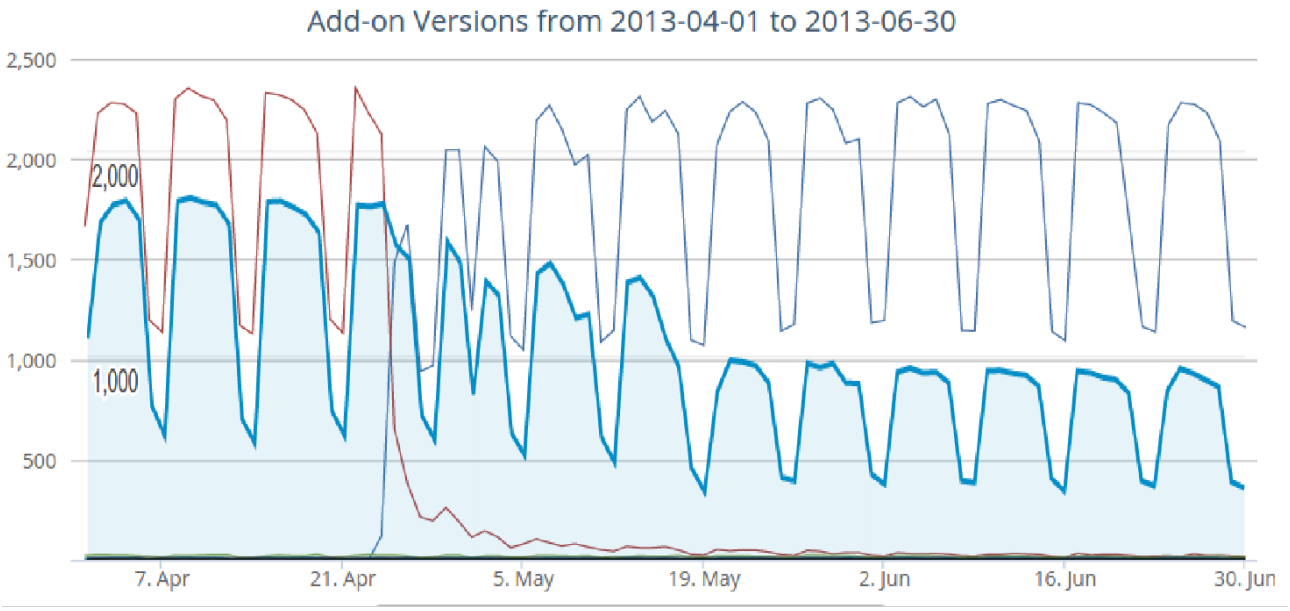


Illustration 38: Version 1.5.0 installation graph

As we can see in the illustration, the installation of this version succeeded without incidents.

Google Analytics drop

As we allowed the user to say “No” to send anonymous data more easily, a lot of users asked to not send data any more. This can be seen clearly on the following graph.



Based on the numbers, the users that were not sending data before this release was a 24% of all the daily users.

After the release this number grew up to a 60%. I had only 40% of the users reporting anonymous data finally.

Reviews effect

This release brought the feature that asks actively to the user to submit a review to Mozilla add-ons site.

Did this action increased the reviews? Let's see.

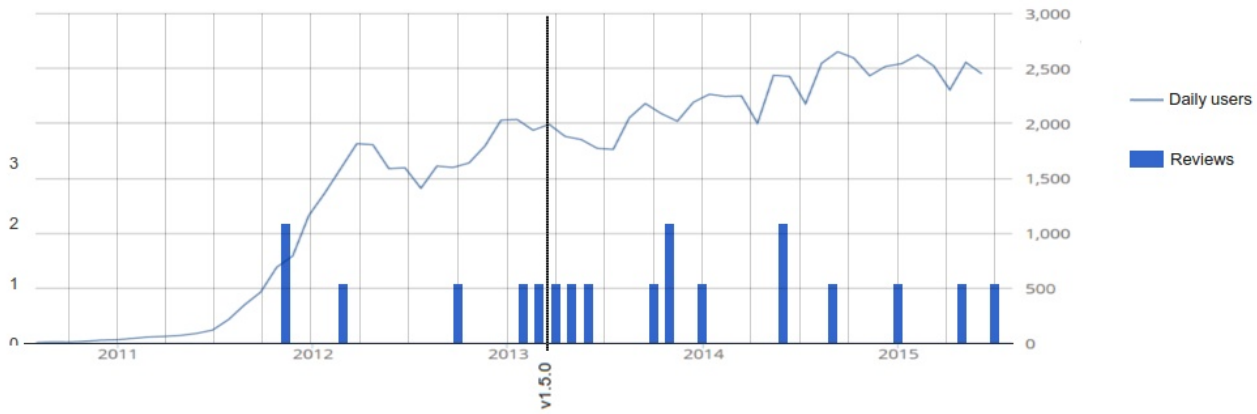


Illustration 39: Active users vs reviews per month graph

There seems to be an increase in number of reviews per month after this release, but the difference is not big enough to conclude that the experiment has worked. It could be because the number of users is bigger or a random effect.

User communications

1. **Bug on Windows 7 + Thunderbird 17**

I received an email from a user that was having issues with Thunderbird and Windows. The extension was giving unexpected errors and was not doing what was supposed, but it did not block the user from using the application. Maybe I should look into this in next iteration.

2. **Bug on Windows 7 + Thunderbird 24.0.1**

This user wrote a 1 star review first and I contacted him to get more information on the bug. He was having the same issue as the one before but with another version of Thunderbird. I did ask him all the needed information to reproduce it, including operating system, add-ons added, and exact flow he was doing and what he expected.

Reviews received

During this period it received several reviews, three of them with five stars and the other was the one mentioned in the last bug above.

Rated ★★★★★
on April 27, 2013

It is wonderful !!!

Very good
Rated ★★★★★
on May 21, 2013

Excellent add-on, works great.
I just wish that when you have a dictionary assigned to an address, you could right click on the message body to temporarily change to another language just for this one message (it happens to me to write in different languages to the same person) without changing the default language.

Thank you for the add-on.

Muy Bueno!
Rated ★★★★★
on June 18, 2013

es un complemento muy útil para aquellos que tienen que escribir en más de un idioma. Y funciona a la perfección.

7.6.4. Conclusions

- Iteration was completed successfully.

- Most users do not want to send anonymous data.

- We do not have enough data to assure that asking for reviews was positive or negative.
- There are bugs to fix for Windows 7
- AMO Certification can depend on the reviewer.

7.7. Iteration 7

The objectives of this iteration were:

- Fix Windows 7 compatibility.

At last the fixes could be resumed as:

1. Some operating systems are slower or initialize components in other order and the users were having problems because some components of Thunderbird were not ready. In this case the spell-checker. The solution was to detect the crash and retry it after half a second.
2. A Windows user also had the dictionary_switcher add-on installed and was interfering with our extension. I had to read the code of this extension to know how it worked and make it compatible.

7.7.1. Testing results

1. All functional tests passed.
2. Tests over Windows 7 were successful.
3. Users that reported the Windows 7 issues tested a beta of the release and confirmed that it was fixed.

7.7.2. AMO Certification results

The certification was successfully obtained and users received the new version.

7.7.3. Community feedback

Beta testers

Before releasing the version to AMO team, I asked the users that contacted me to confirm that the version fixed their issues on Windows 7. Without that it would have been nonsense to publish a new version.

User Reviews

Rated ★★★★★
November 30, 2013

Excellent. The new version works without any problems.

Excellent Idea combined with a bad Idea
Rated ★★★
January 28, 2014

The intended function of this add-on is worth 5 stars indeed.
It works as expected is is very handy for multi language communicators.

BUT: The "collecting usage data" thing is evil - and a bad idea at all. Nothing that is on google statistics is "anonymous". NOTHING.

Rated ★★★★★
June 8, 2014

Great idea and well done implementation.

Everything works as advertised on TB 24.5

Thank you!

Rated ★★★★
June 12, 2014

Excellent add-on, very useful. One star less for info collecting.

Rated ★★★★★
September 25, 2014

Great add-on, very helpful.
Thank you!

Really helpful
Rated ★★★★★
January 4, 2015

Working fine, it really helps me a lot, having correspondents in several countries speaking different languages.

7.8. Installation

The steps to install it once you have Thunderbird are:

1. Start Thunderbird
2. Go to Tools > Add-ons
3. Use upper left search bar and type “automatic dictionary”
4. Press enter.
5. It will show a list of plug-ins including our work:

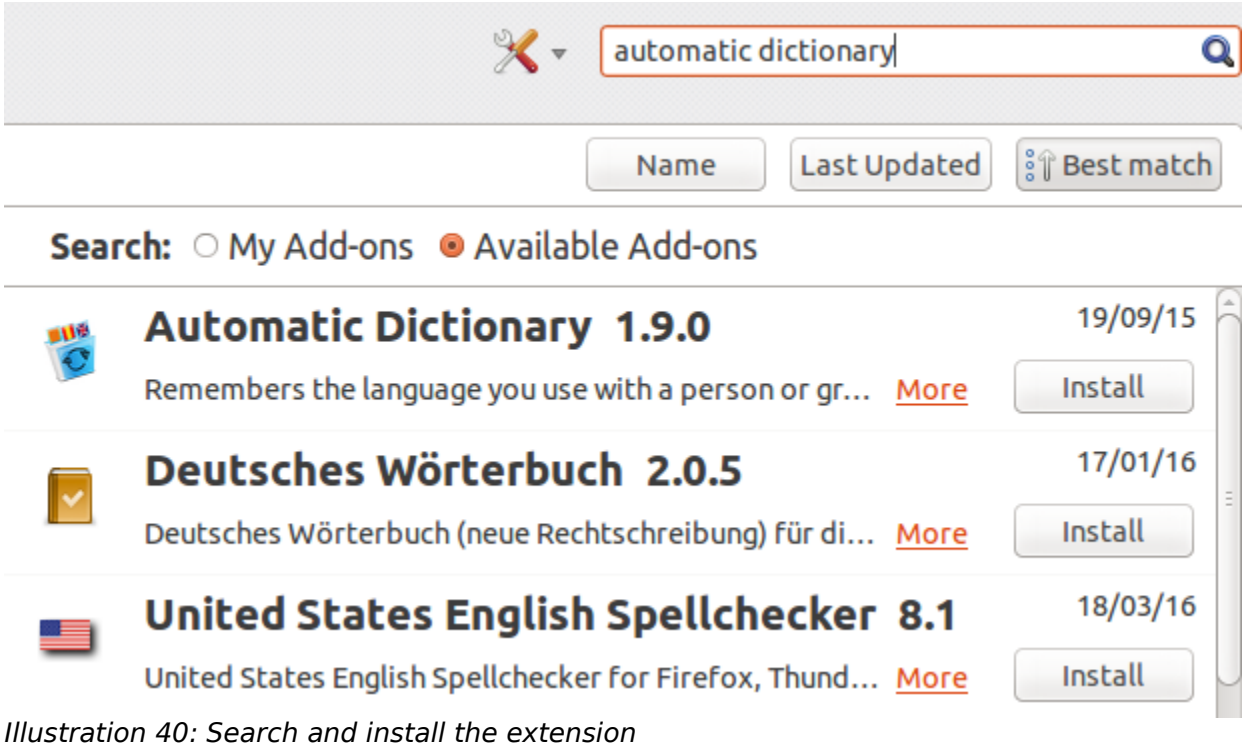


Illustration 40: Search and install the extension

6. Click on install it.

7.9. Usage

To use it, you only have to use Thunderbird as usual. The extension will remember the language you used for that recipients combination and that's all.

Example:

1. You start to compose a message for Alice, but the extension does not know a language for this recipient so it warns you that it have no idea what language to set.

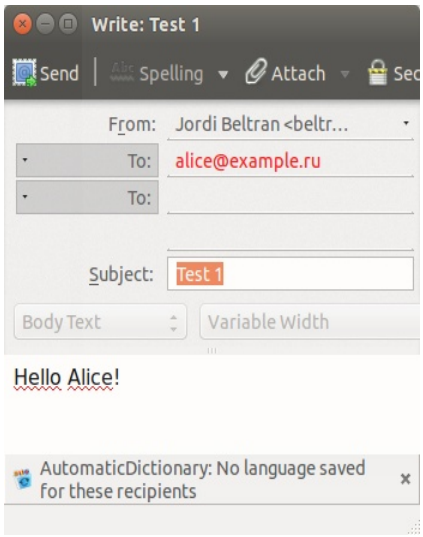


Illustration 41: Usage step 1

2. You can then switch the dictionary to the right language.

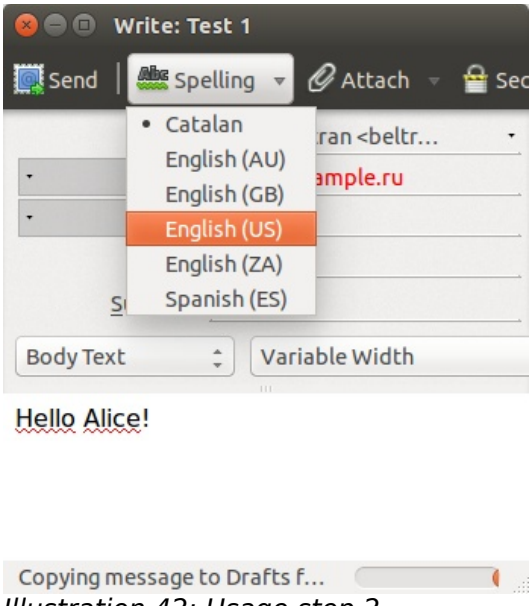


Illustration 42: Usage step 2

3. Then the extension saves this preference for future uses.

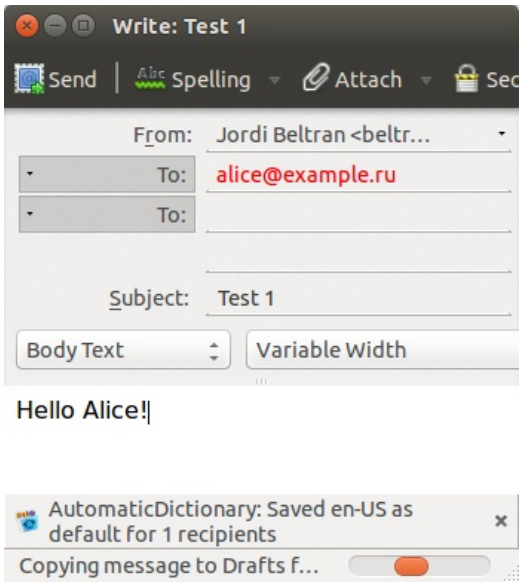


Illustration 43: Usage step 3

So next time you write to this user, the extension will notify that it has set the language defined back then.

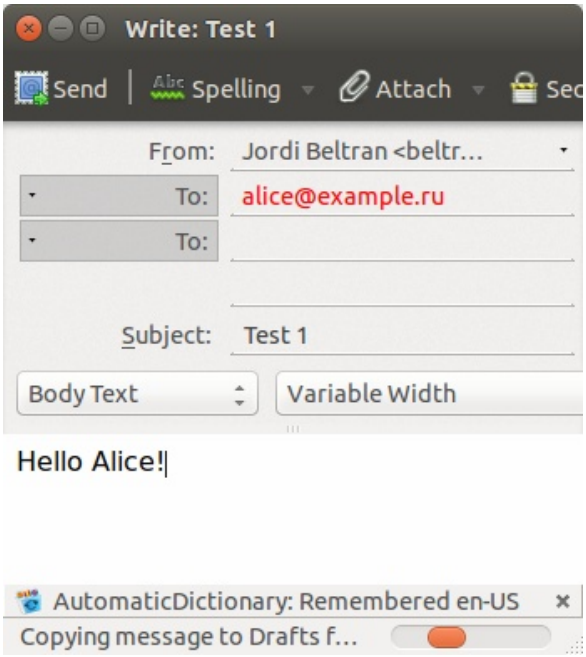


Illustration 44: Usage once saved

8. Conclusions

We can say that the target of this work has been accomplished. The extension has been accepted by the community and it's being extensively used by the users.

There are a lot of users that still do not know this extension, but that is a problem of this type of extension. Users do not know that it exists and they do not look for it.

Most of them find this add-on looking around in the AMO site, and this is something users don't do very often. A lot of them probably will never do it.

Collaborating with this community has been relatively easy, as long as you follow the rules and behave. You need to read and follow the guides, unless you want to have your add-on rejected. AMO reviewers do their job very well and review thoroughly, keeping the add-on community healthy.

Users have communicated with me kindly all the time. Asking for fixes or new features, taking into account that you are doing this for free and altruistically. Some of them even writing some words in Catalan or Spanish to be kind, or asking things about my person or my life kindly.

Interacting with the community and their users has been and is being a real pleasure.

9. Bibliography

1: JUSTINE JORDAN, 53% of Emails Opened on Mobile; Outlook Opens Decrease 33%, https://litmus.com/blog/53-of-emails-opened-on-mobile-outlook-opens-decrease-33

2: Wikipedia, Extreme programming, https://en.wikipedia.org/wiki/Extreme_programming

3: Mozilla, , https://developer.mozilla.org/en-US/Add-ons/AMO/Policy/Reviews

4: Mozilla, XUL - XML User Interface Language, https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XUL

5: Mozilla, Preferences manager API, https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XPCOM/Reference/Interface/nsIPrefBranch

6: Mozilla, SpiderMonkey,

7: Mozilla, Rhino, https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino

8: Mozilla, Marionette JavaScript Toolsw, https://developer.mozilla.org/en-US/docs/Mozilla/QA/Marionette/Marionette_JavaScript_Tools

9: Community, Mocha Test Framework for JavaScript, https://mochajs.org/

10: Mozilla, Mozilla Coding style, https://developer.mozilla.org/en-US/docs/Mozilla/Developer_guide/Coding_Style

11: , JSLint static code analyzer, https://en.wikipedia.org/wiki/JSLint

10. Annex

10.1. Illustration Index