Micro Crowdsourcing: A new Model for Software Localisation

Chris Exton¹, Asanka Wasala², Jim Buckley¹, Reinhard Schäler¹,²
¹Centre for Next Generation Localisation, ²Localisation Research Centre, University of Limerick
www.cngl.ie
www.localisation.ie
chris.exton@ul.ie; asanka.wasala@ul.ie; jim.buckley@ul.ie; reinhard.schaler@ul.ie

Abstract

One obvious flaw in the concept of the knowledge society is our collective failure to date to provide equal access to information and knowledge across languages. We are a long way away from the ideal world, where, as Muhammad Yunus, winner of the 2006 Nobel Peace Prize said, there would only be one language in the information technology (IT) world - your own (Yunus 2007). While the US$16b mainstream localisation industry likes to see itself as the vehicle that is removing this barrier to universal access to digital knowledge and information (i.e. language), in reality it is making limited impact on the widening gap between the information rich and the information poor.

Crowdsourcing has been described as an approach to address the shortcomings of current mainstream localisation, allowing the localisation decision to be shifted from large corporations to service users, thus making IT available in more languages. This paper proposes and describes a new model of crowdsourcing which may provide a platform by which the "equal access to information and knowledge" might be achieved.

Keywords: localisation, digital divide, micro crowdsourcing, real time localisation

Introduction

Current software localisation efforts are largely driven by the economic imperative of short-term return on investment. The localisation decision, i.e. the decision on the languages and locales to be covered by the mainstream localisation effort, is almost exclusively determined by the size of the market a language or a locale represents. Therefore, software is translated for the approximately five million speakers of Danish, but not for the 27 million speakers of Amharic, the national language of Ethiopia. However, there is also a long-term return on investment issue to be considered for commercial organisations. Digital publishers have recognised that, without pursuing deployment in large but currently unviable locales, their product will have limited exposure that may jeopardise future, larger market gains when those locales develop economically.

In addition, the social effects of this short-term economic imperative are grave. Access to information technology is restricted to those speaking the languages of the global north while excluding those speaking the languages of the global south. The majority of people living on this planet cannot share their knowledge in the digital world, nor do they have access to existing knowledge. Organisations engaged in localisation activities not primarily for commercial but for social, cultural, political or developmental reasons, see crowdsourcing as a mechanism to connect with their communities, through reduced cost.

The concept of crowdsourcing was first described by Jeff Howe in his now famous article in Wired Magazine in 2006 (Howe 2006). He described it as the harnessing of a community/group of people to perform a task traditionally undertaken by employees. Crowdsourcing has been taken up enthusiastically and has resulted in almost 8.2 million hits in Google's search engine. It has featured as a major topic at recent, seminal localisation-industry
Localisation Focus

The International Journal of Localisation

events, such as Localization World 2009 and LRC XIII. This is because both commercial and altruistic organizations see it as a mechanism to lower the cost of localisation, enabling them to enter currently inaccessible locales (Rickard 2009). In addition, (Losse 2008) in her keynote at the LRC conference, stated that organizations like Facebook pursued crowdsourcing because it also produced higher quality translations.

Consequently, attempts by companies such as Microsoft, Facebook and Google to create crowdsourcing frameworks that allow volunteer translators and localisers to translate digital content into marginally commercial languages are seen by the industry as having delivered very promising results.

While commentators seem to agree that the main issues around crowdsourcing in the localisation space are control, quality and motivation, there is still a lack of comprehensive studies on any of these issues. In addition, another central issue of crowdsourced localisation, i.e. the need for the localisation decision to be shifted from multinational corporations to the user (Howe 2006), has only been raised on occasion and not as a central pre-requisite for the success of any crowdsourced localisation effort. (Schäler 2008).

Based on these issues, this paper considers how the practices and experience from the open source and Web 2.0 communities could provide a path for software localisation to make Muhammad Yunus’ dream a reality.

1.1 The Cathedral and the Bazaar

The "Cathedral and the Bazaar" is an essay by Eric S. Raymond on software engineering methods (Raymond 1999), based on his observations of the Linux kernel development process and his experiences managing an open source project. In it he describes how the traditional software development paradigm could be viewed as hierarchical and tightly planned; Raymond likens this view to a Cathedral which is monolithic and obviously architectured by some controlling authority. The open source development paradigm however, he continues, could better be likened to a market or bazaar, where it is obvious that industry of some kind is occurring but there seems to be no or little central authority or control.

Traditionally, mainstream software localisation has been tightly controlled by multinational corporations. They strictly managed everything from the localisation decision itself to the selection of an appropriate localisation process, the use of certain terminology and translation memories, and the deployment of adequate tools and technologies.

This model is not unlike the Cathedral model described by Raymond, with its central control and tight management through a number of levels of activity and quality control, to ensure a suitable and tested final product. Indeed, the type of software often used to support and control the activity of localisation, (essentially customised project-management systems such as Idiom Worldserver) provides strong evidence of the approach chosen by mainstream localisation. However, such a 'Cathedral' model brings with it the implicit need for large coordination efforts and subsequently high costs.

The 'Bazaar' model, in turn, is associated with the open source community, and requires lesser controlling authority. Work is somehow carried out in an almost chaotic, community-driven manner. This has provided the business and technical communities with a suite of software, including Linux (Raymond 2001), Apache (Mockus et. al 2000) and Openoffice (Feller and Fitzgerald 2002), upon which many companies rely heavily today. These systems are proof that a community-driven, open source model can also deliver quality software systems.

Indeed the open source community model may provide a paradigm to address some of the problems faced by the localisation community in relation to their desire to expand into underserved markets and to break down the digital divide. However, many initiatives addressing underserved markets today, such as the ones initiated by Facebook, Microsoft or Symantec, still have a central authority driving the localisation effort, rather than being bottom-up and community-driven.

This paper proposes an alternative approach to the idea of crowdsourcing in relation to the translation of software systems. In this model, individual users translate elements of a system and its documentation as they use them in return for free access to these artifacts. Periodically, the elements of the system and documentation translated by the individual translators are gathered centrally and aggregated into an integral translation of all, or parts of, the system.
2. Approach to Micro-Crowdsourcing

Consider a software package such as Open Office that has been developed for a purely English speaking audience. Even if this product were designed to facilitate its easy adaptation into other languages it would still require the effort of either a number of altruistic individuals or the coordinated effort of expensive professional localisers to make this product available in another language.

Where there is no/limited immediate economic imperative for the digital content publishers, such as in the case of open source software or voluntary organisations aiming to bridge the digital divide, one solution might be the automatic translation of content into non-commercially viable languages. Although this option might be preferable to simply ignoring these languages, automatic translation is not yet at a stage where such a product could be released with confidence.

Imagine however, developing a software application such as Open Office that allowed a community of users to update the user interface in situ, either directly from the original English version or perhaps working from a less than perfect machine translation. The update could be enabled via a simple popup micro localisation editor that would allow them to change UI text in situ simply by ctrl-clicking on any text that is displayed.

This editor may have to enforce constraints on the translation, such as restricting string length, and could perhaps include appropriate translation memories and standards to assist in the translation. As a ctrl-click could be applied to any displayable text area, error messages and help information messages could also be included as translatable material. Indeed, the editor may even go as far as allowing graphical replacement of certain artefacts.

The result would be a set of textual (and possibly graphical updates for each user. Then suppose that each update-set could be automatically gathered in a central repository that would, in turn, push update events back to the community of users, periodically or on-demand. This would update their product with the latest translations. Imagine that these users could, in turn, quality assure the updates and re-instigate the cycle, in the same way that Web 2.0 communities like Wikipedia reach consensus by iterative refinement.

Such an approach would represent a radically novel approach to localisation requiring a novel architecture, as demonstrated in figure 1. On the far left, we see a central server that receives and sends updates to and from individual deployments of a

![Figure 1: The UpLoD-Based Architecture for Micro Crowd-Sourced System Translation](image)
software package called 'Writer', one of which (to the left of the figure) is substantially expanded.

As can be seen from the 'expanded Writer', the three tiers of the application are augmented by an 'Update-Log-Daemon' (UpLoD) module. This UpLoD module allows the user to update the user interfaces as they use the system and log the changes in a local audit file. The records in this local audit file contain unique identifiers for the GUI elements that have been changed. The identifiers are associated with the pre-translation and post-translation. Periodically, a Daemon trawls the audit log and, on finding new records, passes updates to the central repository on the server.

These updates can be handled in a number of ways. For example in publically edited wikis, revision control enables a human editor to reverse a change to its previous version. For a "Micro Crowdsourcing" system it is possible to consider that there might be a limited number of trusted editors (self moderating) for a specific language group to tidy up the localisation in this fashion. A version control system would then enable editors to build a release package on a periodic basis based on the influx of micro changes from standard users. This would serve to drastically decrease the number of changes and updates to the UI and avoid updates with new translations on an ongoing basis.

For a more automated approach the server might periodically analyse the update set of all users, based on an aggregate consensus, and may be to recommend the changes to be made to the other versions of 'Writer'. These changes are captured by the UpLoD module and update the GUI correspondingly.

Another open source development concept which could be adapted to suit the "Micro Crowdsourcing" model is that of distributed revision control. Distributed revision control is built on a peer-to-peer approach, unlike the centralised client-server approach classically used by software versioning systems such as CVS. In a distributed revision control system each peer maintains a complete working copy of the codebase. Synchronization is conducted by exchanging patches (change-sets) from peer to peer, a more in-depth discussion of the process is described by Noah and Adam (2009).

Regardless of the version control system that will be used, the translation is carried out in an incremental, ad-hoc manner by a community of (not necessarily experienced) "translators", each of whom would double as a proof reader for each other's work.

Once we allow all registered end users to become translators or localisers, we spread the workload over a large user base. The limiting factors would be the number of bi-lingual speakers with access to computers and internet connectivity. However, even this limiting factor could be overcome by offering monoglot users, familiar with the software, access to suitable translation aids including machine translation, translation memories and terminology databases.

To a large degree, a similar model already exists in the Wikipedia community where content may be added and amended by any registered user. Quality and precision, issues discussed as highly problematic in the context of crowdsourced localisation, are in this case simply promoted by the fact that any reader of Wikipedia can register and thereby correct or update any particular entry. This phenomenon can be likened to the "many eyes" principle associated with open source. This phrase was coined by Linus Torvalds (Raymond 1999) states "Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix will be obvious to someone." It simply describes the notion that, since open source code can be viewed and potentially changed by anyone who cares to look at it, the number of bugs that are caught and fixed increases dramatically compared to proprietary driven development. Likewise, it is envisaged that this "many-eyes" characteristic of the UpLoD architecture will promote an increasingly stable, high quality, and locale-specific application over time as users are empowered to become part of the localisation process.

In addition, the software is translated for free by volunteers, provided that the digital publisher is willing to deploy its un-translated or automatically pre-translated version to registered volunteers in each locale. In open source scenarios and scenarios where the aim is to bridge the digital divide, this will not be a concern. However, even in commercial scenarios the corporation may consider the exposure gained from having a localised version to be worth the loss of potential license revenue from registered volunteers. This would be particularly true in the case of emerging markets where the possibility of sales might be low at the moment, but where early exposure to localised systems could lead to commercial opportunities later.
3. Proof of Concept Implementation

A proof of concept prototype of this architecture was created to validate and refine this approach. The prototype consists of two components: the central server component and a simple RTL (Real Time Logging) Notepad application which imitates the "Writer" of Figure 1. The UpLoD module was implemented and integrated in the RTL Notepad application in addition to its generic text editing functions. Due to its simplicity and portability the Portable Object (PO) file format was chosen as the format for the local audit file.

In the RTL Notepad, simply right-clicking on text inside any UI element brings up a context menu where users can enter into a 'localisation dialog' (See figure 2). From this window, a user can translate the selected UI strings. The changes are reflected in the UI in real-time. Options have been provided to users in the localisation dialog for the online transmission of their changes to the central server or for the offline saving of the translation to the local audit file for later batch transmission to the server.

In this prototype, we propose an automated translation voting mechanism to ensure the quality of the translation. For this purpose, the server maintains a database containing translations and the number of votes for each translation (ie: the number of users who have suggested that translation), for each language.

In the following sections, the main phases of the localisation process associated with this architecture are explained in more detail. For illustration purposes, screenshots of the RTL Notepad in English and its translated version in Sinhala are given in Figure 3 and Figure 4.

3.1 Initialisation

The RTL Notepad application can be configured to update its UI by connecting to the central server or by reading from its local audit file, i.e. in line with the changes made by its immediate user. If the Notepad is configured to connect to the central server, it will retrieve the translations from the server and will update its user interface accordingly. This may result in overwriting the customisations already made by the user and they should be alerted to this possibility before they agree to the update or they should be alerted on a (UI) string-by-string basis.

In the first scenario, i.e. when the RTL Notepad is configured to update its UI using the information obtained from the server, the RTL Notepad will send an HTTP request to the central server stating the user’s language, as configured in the RTL Notepad application. Then, the server will send an XML response which contains a list of source-target translation units for all the UI elements. This process is illustrated in figure 5. The server will generate the XML by choosing the translation with the highest number of votes, for each UI string. The RTL...
Notepad will process the XML response and update its UI. An illustrative XML response is given in figure 6 (where the GUID tag uniquely identifies the UI element).

There will be situations where different translations for the same UI string have the same number of votes. To handle such potentially ‘thrashing-like’ scenarios, the RTL Notepad will show a special dialog, allowing users to choose their preferred translation during start-up. In order to minimise such translation conflicts in the future, the user preferences are sent back to the server so that the server will increase the votes for the relevant translations. They will also be stored locally so that the user does not have the repeat his/her choice.
3.2 Translation Submission Process

Users can submit translations of UI strings to the central server through the localisation dialog of the RTL Notepad. User submissions will be directed to the central server as HTTP requests. Upon submission, the central server will query its database to see whether the submitted translation already exists. If so, the server will increase its votes. Otherwise, the translation will be added to the relevant language table, initialising its number of votes to one.

3.3 Community Suggestions Retrieval Process

In the localisation dialog of RTL Notepad, an option is given to retrieve the suggestions of the user community. Once the 'View other suggestions by community' option is selected in the RTL Notepad, it will send an HTTP request to the server asking for community translations for the selected UI string. The server will send an XML response to the client. The UpLoD module in RTL Notepad will process this XML and list these suggestions in its localisation dialog, in the order of number of votes received, as illustrated in figure 2. The users then can choose their preferred suggestion to be used in the GUI of that version of RTL Notepad. Once a user selects a suggestion, the suggestion will be sent back to the central server as an HTTP request. The server will then increase the votes of the given suggestion.

4. Outstanding Challenges

Of course, localising software is not as simple as portrayed in this prototype. Not only does text have to be changed: holding boxes have to be resized, and images may have to be replaced, for example. However if tools that facilitate localisation were incorporated into an UpLoD-type architecture, it would not be unreasonable to expect that the need for these changes could be covered satisfactorily. After all, such changes are currently performed in existing localisation efforts. The model proposed here suggests piggy-backing such functionality into the "Update" component of the system deployed.

There is also the possibility that volunteer translators would focus their efforts on only a small proportion of the user interface. This proposition is based on Pareto's Principle (Bookstein 1990) which, to paraphrase for this context, suggests that most users of large applications will only use a small proportion of its functionality. If translators choose to translate as they use, or choose to do the translations that others will see, rather than translating holistically, it is likely that translation coverage will be patchy and will result in a 'pidgin' system made up of translated 'frequently-used' facilities and untranslated 'infrequently-used' facilities. This may prove sufficient for the majority of users, but runs the risk of frustrating users who have more demanding requirements. However, frustration can become a motivating factor if the user is empowered to subsequently change the associated UI strings.

Another potential challenge is that the voting
mechanism proposed may prove insufficient and ineffective; specifically, there is the possibility of 'thrashing', where two individual translators, or groups of translators, have very strong and conflicting ideas about the translation required for specific GUI elements. In such instances, the 'Analysis Engine' of the central server would need to intervene, analysing the central logs, deriving the appropriate translation, possibly with human intervention, and locking future changes.

Indeed, we see this 'thrashing' problem as being one of the main issues with this approach. Imagine, as a user, you customise the interface and then send your changes to the server. Imagine then, retrieving the server-side customisations and finding that very few of your changes had survived. This is a micro-form of thrashing that would probably be prevalent, particularly if there were a wide number of users customising the interface - a measure of the approach's success. Such negative feedback might discourage the user from making further changes to the interface and result in a fall off in localisation activity over time. Indeed, it might discourage them from using the application itself, as the interface they strove to create has been destroyed by the server-side customisations. Hence, as mentioned in section 3.1, we see a strong role for 'change alerts' and the option to opt out of server-side customisations as core for the users of this approach.

The voting mechanism currently implemented in the prototype takes no account of user quality, an attribute that could easily be calculated from the available data (a simple measure could be the percentage of each user's suggestions that equate to the customisations with the highest vote). This additional information could be used to resolve ties, where equal numbers of votes were obtained for two or more different translations, to resolve thrashing or, more generally, as a weighting on the votes. It may be that user submissions might have to be reviewed by human experts (preferably by a pool of linguists) prior to committing to the server's database. This additional step would ensure the quality of the translations to be used in UI elements in terms of criteria such as relevancy, accuracy, suitability, and consistency. However larger scale deployments, where a bigger community is involved, may well counteract this potential issue.

It is noteworthy to mention the programming difficulties that may be encountered when developing UpLoD-architecture-based applications. The development of the prototype revealed that some UI widgets and built-in UI components such as file open dialog, printer dialog etc. provided by several programming languages are tightly integrated with the underlying platform and hence cannot be modified to incorporate real-time localisable features.

Notwithstanding, UpLoD-architecture-based applications are easy to develop using programming languages that support object oriented programming (OOP), especially if their UI development components are loosely coupled with the operating systems and the rest of the system's components. Indeed, ideally, the "real-time localisability" should find support within the operating systems and the programming languages themselves.

However, as long as this is not the case, it has to be acknowledged that there is an overhead associated with the UpLoD architecture that adds to the expense of this initial development, so ideally this architecture should be as pluggable-and-playable in nature as possible, and this is seen as an area of future research for the group.

This overhead may be increased in a large-scale UpLoD system. For example, if the deployment was wide enough, server farms may have to be designed for load balancing as well as efficient processing of client requests.

It would also be interesting for future research to investigate the possibility of using the UpLoD architecture for the localisation of existing applications. One possibility is to develop a daemon or Windows service that would facilitate this. The daemon or the service could display translations as tooltips whenever the mouse is hovered over the UI strings of the existing application.

Future work will include the development of a suitable light-weight localisation model that includes an appropriate container that could facilitate a new and ongoing micro versioning capability. To accompany this a micro versioning workflow model would have to be developed that could facilitate and address many of the features described throughout this paper, for example the capability to facilitate a 24 hour micro update capability that could cover up to 100+ languages on a 24 hour basis.

Appropriate techniques for the development and maintenance of an associated translation history.
would also be a major objective. It is envisaged that these issues will be worked through, by the development of a series of prototypes for a selected sample open source application and associated user trials. This iterative design approach will then serve to inform on the overheads required to implement the UpLoD architecture and drive development of the associated tools and facilities required to optimise this approach.

References


