Making Science Simulations and Websites Easily Translatable and Available Worldwide: Challenges and Solutions

Wendy K. Adams[1], Hisham Alhadlaq[2], Christopher V. Malley[3], Katherine K. Perkins[4], Jonathan Olson[4], Fahad Alshaya[2], Saleh Alabdulkareem[2], and Carl E. Wieman[4, 5]

[1]*Department of Physics, University of Northern Colorado, Greeley, Colorado 80639, USA*

[2]*The Excellence Research Center of Science and Mathematics Education, King Saud University, and the Department of Physics and Astronomy, College of Sciences, King Saud University, PO BOX 2455 Riyadh, Saudi Arabia*

[3]*PixelZoom, Inc., Boulder, Colorado. USA*

[4]*Department of Physics, University of Colorado, Boulder, Colorado 80309, USA*

[5] *The Carl Wieman Science Education Initiative, University of British Columbia, Vancouver, British Columbia,V6T 1Z3, Canada*

Corresponding Author: Wendy K. Adams, wendy.adams@colorado.edu, 970-539-6154, Fax: 970-352-3506

Abstract

The PhET Interactive Simulations Project partnered with the Excellence Research Center of Science and Mathematics Education at King Saud University with the joint goal of making simulations useable worldwide. One of the main challenges of this partnership is to make PhET simulations and the website easily translatable into any language. The PhET project team overcame this challenge by creating the Translation Utility. This tool allows a person fluent in both English and another language to easily translate any of the PhET simulations and requires minimal computer expertise. In this paper we discuss the technical issues involved in this software solution, as well as the issues involved in obtaining accurate translations. We share our solutions to many of the unexpected problems we encountered that would apply generally to making on-line scientific course materials available in many different languages, including working with: languages written right-to-left, different character sets, and different conventions for expressing equations, variables, units and scientific notation.

Keywords: Simulations, science education, research, languages, websites, online resources, translation

Introduction

As a part of the PhET Interactive Simulations Project a method was created for translating the PhET science and math simulations and then hosting them on the PhET website (http://phet.colorado.edu). This approach works because the underlying science is universal and it is only the words and norms used to represent it that must be translated. In this paper we present the current translation process and discuss the pitfalls we encountered along the way. By documenting the process, we hope others can avoid many of the difficulties we encountered while creating such a tool.

The PhET Interactive Simulations Project is a substantial and growing suite of professional quality simulations (currently ~90) for teaching and learning science. The simulations are written in Java or Flash and are distributed from the PhET website at *no cost* to users, with roughly 10 million uses in the past year. The majority of PhET simulations are for teaching physics, but there are a growing number in chemistry, biology, math and other sciences. PhET simulations provide a high degree of interactivity in terms of user control, dynamic feedback, and multiple representations. The simulations enable students to make connections between real life phenomena and the underlying science which explains such phenomena (Fig. 1). The PhET team of scientists, software engineers and science educators uses a research-based approach – incorporating findings from prior research and internal testing – to create simulations that support student engagement with and understanding of scientific concepts. The PhET project's research includes investigating the use of PhET simulations in a variety of educational settings. (PhET Interactive Simulations 2009).

This paper describes the current solution for translating the simulations and the PhET website including the technical issues involved. It also explains the method that was settled on for finding translators and finally, co-authors from the Excellence Research Center of Science and Mathematics Education (ERCSME) at King Saud University share their experience as users of the Translation Utility. Creating this method for translating simulations was a difficult and time consuming process; however, it was much more efficient than leaving it up to various groups to rewrite the simulations from scratch. It also allows a single location to host the same product in multiple languages, giving the instructor the opportunity to choose which language s/he will use in their classroom.

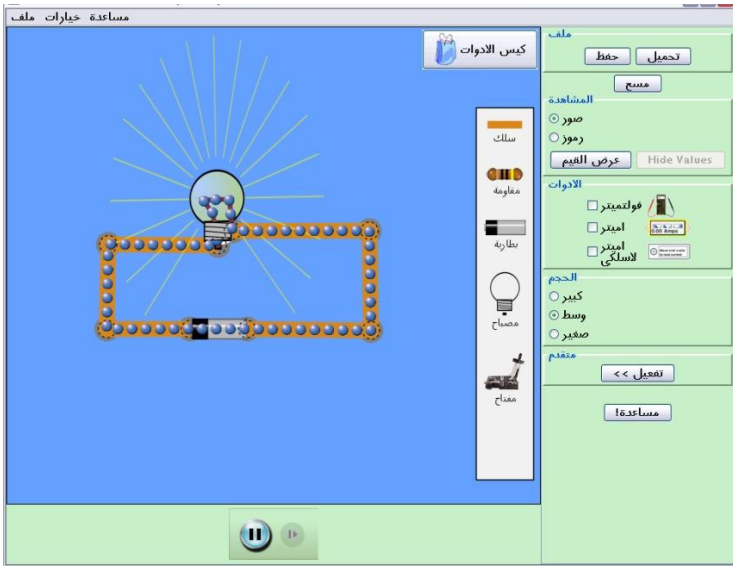

Fig. 1 shows a screen shot of Circuit Construction Kit translated into Arabic. This simulation allows the user to create circuits using batteries, light bulbs, switches and wires. When a circuit is completed, the light bulb lights.

Current Solution

The PhET project has several underlying program goals that constrain the possible approaches to translating simulations. PhET must remain free, be easily accessible to teachers, scientifically accurate and user-tested. In pursuit of these goals PhET has multiple delivery modes (online, offline, entire website offline), and the online delivery requires only the click of a mouse to launch a sim (no certificates). In addition with PhET having simulations in both Java and Flash, it is cost effective to have similar translation solutions for both Flash and Java sims.

In December 2007 the Translation Utility for PhET simulations was released (PhET Team 2007.) At the time of writing this paper the Translation Utility had been available for 30 months and PhET has posted 1,758 translations of simulations. The Translation Utility is a program that allows the translator to see the English strings contained in a selected simulation in one text box as they enter their translation in an adjacent box (Fig. 2). The translator can then test their work in the actual simulation to make sure it appears as intended. Once complete, the translator emails the translated strings to PhET. PhET then compiles a translated version of the simulation and makes it available on the PhET website. This solution came after several other avenues for acquiring translations proved unsuccessful. In addition, it has recently become obvious that the delivery mechanism, the website, should also be available in the user's preferred language.

Translators

PhET first tried to use Google Translate to translate the website and simulations into Spanish which often resulted in incomprehensible, or highly inappropriate, Spanish phrasings. Next PhET tried hiring bilingual students, but it turned out to be very difficult to find students with the necessary science background. Neither of these approaches led to a high-quality product and neither had the potential to lead to translations in 51 languages.

The most successful method for translating the PhET simulations turned out to be a contributor model similar to the Wikipedia editing model[1]. Instructors who use PhET simulations volunteer their time and expertise to create translated simulations. These instructors are ideally suited for creating the translated versions. On the surface this contributor model appears to be the least expensive; but, there are much more fundamental advantages. Bi-lingual teachers who use the simulations have three necessary characteristics: They know the scientific terms and notation in both their language and English; they are using the simulation in class so are familiar with the content; and they have a personal interest in the final product because they need to use it in their course(s). An example of why the translator needs to be a teacher familiar with the content appeared in a Russian translation of Battery-Resistor Circuit. "Show core" was translated as "Show nuclei" where the use of "nuclei" is scientifically the incorrect concept.

PhET has encountered one other group that meets these requirements who were not actual instructors. A teacher in Romania created a class project where his students each translated a simulation and entered it in a contest. These translated simulations turned out to be of very high quality since the students were in the unusual situation of knowing the simulations, knowing the science of both languages, and being motivated to produce a high quality finished product.

[1] Wikipedia's editing model is not to be confused with their translation model which has fully independent entries.

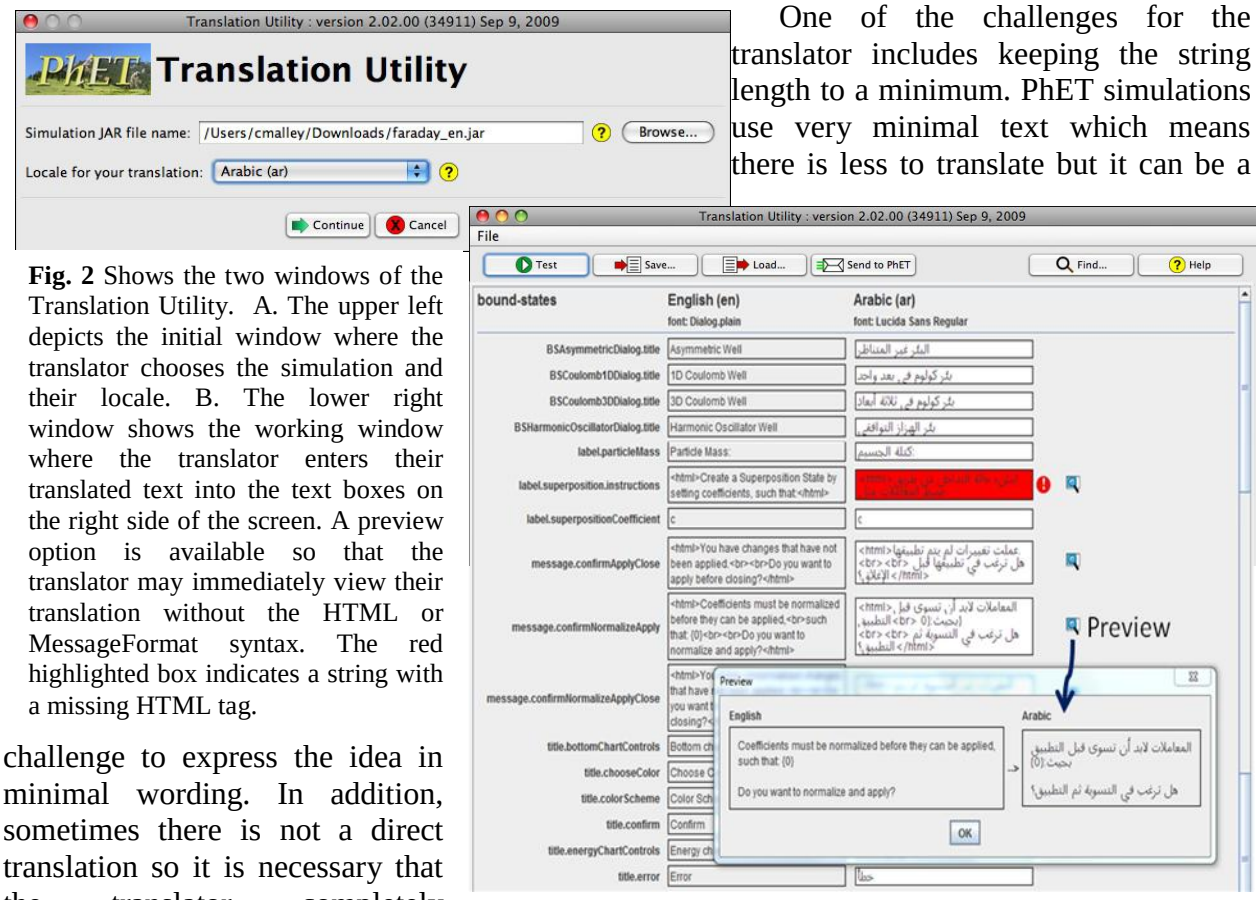

Fig. 2 Shows the two windows of the Translation Utility. A. The upper left depicts the initial window where the translator chooses the simulation and their locale. B. The lower right window shows the working window where the translator enters their translated text into the text boxes on the right side of the screen. A preview option is available so that the translator may immediately view their translation without the HTML or MessageFormat syntax. The red highlighted box indicates a string with a missing HTML tag.

One of the challenges for the translator includes keeping the string length to a minimum. PhET simulations use very minimal text which means there is less to translate but it can be a challenge to express the idea in minimal wording. In addition, sometimes there is not a direct translation so it is necessary that the translator completely understand the material and context of its use, which is one reason the translator usually has to be a teacher.

Simulation Translations

It may appear that the contributor model is a cost efficient option; however, high level professional software developers are quite expensive, and it took them 180 hours to create the Translation Utility in addition to the time spent internationalizing each text string in the simulations. Much of this time spent developing the Translation Utility was figuring out the most efficient design.

Developing software for world-wide use involves addressing three related issues: *locale specification, internationalization* and *localization*. In computing, a *locale* is a set of parameters that defines the user's language, country, and any special variant preferences that the user wants to see in their user interface (Deitsch 2001). For PhET simulations, a locale is defined by a *language code* and an optional *country code*. *Internationalization* is the process of designing software so that it can be adapted to various locales without engineering changes (Deitsch 2001). Internationalization is often abbreviated as i18n (there are 18 characters between 'i' and 'n'). *Localization* is the process of adapting internationalized software for a specific locale (language and country) (Deitsch 2001). Below we describe what we learned about locale specification, i18n, and localization as related to simulations that would be of use to other groups with similar goals.

Locale Specification

There are several different standard conventions for specifying a locale, and the first decision is which convention to use. PhET specifies a locale using codes defined by the International Organization for Standardization (ISO). ISO 639-1:2002 defines language codes (International Organization for standardization 2007), while ISO 3166-1 defines country codes (International Organization for standardization 2010). While there are other standards, these are the most commonly-used definitions. An example of a language code is pt_BR; the language is Portuguese (pt) and the country is Brazil (BR), separated by an underscore. The country portion of the locale is optional for languages where localization does not differ by country.

PhET's early attempts to specify locale used only a language code. Feedback from translators quickly made it clear that language code alone is inadequate; regional differences often require different translations for the same language. For example, Chinese (language=zh) requires different translation for China (country=CN for Simplified Chinese) and Taiwan (country=TW for Traditional Chinese). So we need to be able to specify zh_CN and zh_TW as separate locales.

Internationalization (I18n) of Simulation Software

I18n support is part of PhET's common framework- software code that is shared by all simulations. Additional i18n support is also provided by Java and Flash.

The first task of i18n is to read and interpret the locale. For Java, the most common method of reading a locale is via standard system properties that identify the locale to the Java Virtual Machine (JVM), which enables Java's built-in i18n support. For Java sims delivered via Java Web Start (JWS), we do not have permission to set the locale for the Java Virtual Machine, and are therefore unable to use much of Java's i18n support. (We are investigating signing JAR files with digital certificates – see discussion under *ongoing challenges* below. Doing so would allow us to set the locale for the JVM and take advantage of Java's i18n support.) Instead of using standard Java system properties, we use PhET-specific properties to specify the locale, and provide our own implementation to read and interpret the locale.

Flash handling of locales is similar to Java. Flash's built-in locale support only identifies language, and is not capable of identifying country code. So we have implemented our own mechanism (using FlashVars) to specify and interpret the locale. As with Java, this prevents us from taking advantage of Flash's limited i18n support for rendering fonts.

Once a locale has been identified, a simulation must decide what needs to be adapted to fit that locale. Many things can vary by locale, including language (e.g., alphabets, fonts, writing direction), writing conventions (e.g., date formats, number formats, equations) and cultural differences (e.g., symbol meanings, significance of colors).. PhET simulations currently support language-related differences; other differences are not generally supported, or are supported on a sim-by-sim basis. Here we will look at how PhET simulations support i18n of strings (collections of characters from a specific alphabet), fonts, and writing direction.

Strings are internationalized by replacing literal strings with symbolic key values. Given a locale and a key value, a lookup is performed at runtime, and a locale-specific string is returned. If no locale-specific string is found, the default (English) string is used. For example, code that is not internationalized might look like this in Java (Flash code is similar):

```
String playButtonLabel = "Play"; // a literal string
```

Internationalized code would look like this:

```
String playButtonLabel = TranslatedStrings.lookup( locale,
                "button.play" ); // a lookup
```

Translated strings are stored in locale-specific *string files*. For Java simulations, string files are Java properties files; for Flash simulations, they are XML files. (Java and Flash natively support different file types.). String files contain key/value pairs, with the values using characters from the Unicode Standard (The Unicode Consortium, 2009). In Java properties files, all characters that are not in the Basic Multilingual Plane are in an escaped form, while the characters in Flash string files are directly encoded in UTF-8.

The Java string file entry for the example above would look like this:

```
button.play=Play
```

And the Flash entry would look like this:

```
<string key="button.play" value="Play"/>
```

Each simulation has its own set of string files, one file for each supported locale. Strings that are common to many sims are stored in a special common-strings file, so that they only need to be translated once.

Once we have a translated string, it must be rendered using an appropriate font, and in the proper writing direction (left-to-right, right-to-left, or mixed direction). All PhET sims rely on the user's computer to provide fonts. To select an appropriate font in Java sims, PhET maintains a table of preferred fonts. If a preferred font is specified for a locale, and that font is found on the user's computer, then that font is used to render strings. If no preferred font is found, then the default font on the user's computer is used. The default font may or may not be appropriate (e.g., fonts that do not contain Japanese characters are inappropriate for Japanese translations). If an inappropriate font is used, strings may be rendered improperly. In both Java and Flash simulations, for example, characters that cannot be rendered properly appear as empty rectangles. It would be very difficult to include appropriate fonts for each language and would substantially increase the file size for sims (by ~1.5 MB for *each* font). Thus, PhET does not attempt to distribute fonts with simulations, and relies on translators and users to report font problems and to assist in identifying preferred fonts.

For Java simulations, the writing direction of strings is handled automatically by Java's text renderer. Based on the specific Unicode characters, the renderer assembles the characters in left-to-right, right-to-left or mixed direction. Flash lacks the native ability to handle writing direction, and instead detects and uses the default writing direction of the user's computer. For example, a Windows computer without right-to-left language support will display Arabic characters in an incorrect left-to-right direction; however, after support is properly installed, both Arabic and English will be displayed correctly.

Some strings contain *placeholders* that cannot be filled in with text until runtime. These strings present additional writing-direction issues that must be addressed. For example, consider the string "__ is starting up." where the blank will be filled in with the simulation's title. To give the translator control over the order of the words in these types of strings, PhET uses Java's MessageFormat syntax (Oracle 2010) to specify parameterized strings in both Java and Flash. In English, the above string would be "{0} is starting up." Placeholders are specified as {0}, {1}, {2}, etc. and are filled in by the program at runtime. A pitfall of this approach is that it is sometimes difficult for the translator to determine the semantics of the placeholders.

HTML strings also require formatting knowledge that translators may not have, such as tags . To help address both MessageFormat and HTML syntax difficulties, the Translation Utility now includes a preview capability for each individual string (Fig. 2B) so that the translator does not have to run the whole sim preview to check each string.

Localization and the Translation Utility

Localization is the process of adapting internationalized software for a specific locale by adding locale-specific user interface components and translating text (Deitsch 2001). PhET localization is limited to translation of strings; we do not provide general support for adding locale-specific components to simulation interfaces. For example, uploading an image as part of a translation is not possible. In some cases, text is in an image (formula in Flash sims) and so cannot be translated.

Translation of both Java and Flash simulations is supported by the Translation Utility (PhET Team 2007). The Translation Utility provides a translator (a person creating a locale-specific translation) with a GUI (Graphical User Interface) for creating PhET string files. Translators choose their locale from a list of locales (Fig. 2A). The GUI then displays two columns: a left column shows the English strings, and a right column allows translators to enter the corresponding strings for their locale (Fig. 2B). A translation can be tested at any time by pressing a "Test" button, which will run the simulation in the specified locale, with the translator's strings. The translator should be checking that their translation is appropriate, is rendered properly, and does not create user-interface layout problems. Layout problems can occur when a translated string is significantly longer or shorter than the original English string. When the translation is completed, pressing a "Submit" button creates a string file, which can then be emailed to PhET.

Submissions are reviewed by PhET, and then published to the PhET website. When PhET "reviews" a submitted translation, it is checked to see that it runs, is complete and the layout is O.K. But, the PhET team is not qualified to check the accuracy of translations. PhET depends on the user population to notify them of problems – similar to Wikipedia's editing model. So far only a few replacement translations have been received with the changes aimed at improving completeness. There have been comments from one translator that three of the existing translations are adequate but not ideal, and happily there have been no objections to any existing simulation translations.

Website

Simulation Delivery

The PhET website offers three possible delivery methods for simulations: 1) Run a simulation online which involves clicking the "run now" button on the website and requires live Internet connection; 2) Download an individual simulation by choosing the "download" option, then the simulation is saved to the user's computer for use at anytime; and 3) Download the PhET offline website. This is an installation program that puts the website and all of its content (except teacher contributed activities) onto the user's computer. A user then has future access to all the simulations and supporting material without the need for an Internet connection.

All languages for each simulation are available on the website. Access can be found either on each individual simulation page or on the "translated simulations" page. To deliver translated

versions of the simulations *online*, Java simulations are run via locale-specific JNLP files, executed by Java Web Start, with translated strings bundled in the JAR file. Flash simulations are run via locale-specific HTML files, with translated strings embedded in the HTML. Flash passes the locale to the Flash Player through FlashVars.

When users *download* an individual simulation, they can choose which language they would like. However, this is done using locale-specific JAR files for both Flash and Java simulations rather than the Java Web Start JNLP files or HTML files that are used for online delivery. Similarly, the PhET offline website includes all the translated versions of the simulations, both Flash and Java as JAR files. These locale-specific JAR files contain string files for all locales supported by a simulation. The *offline delivery* of Flash simulations as a JAR file creates the disadvantage of requiring the user's computer to have Java in addition to Flash. Since no complaints have been received from users, this solution appears to be adequate.

In general, all delivery of PhET sims is locale-specific. When the user selects a sim to run, they also select the desired locale. The sim therefore ignores the locale of the user's computer, and runs in the locale specified by the user. This is different from typical software models, where the application runs in the locale of the user's computer. This difference presents many challenges, and precludes the use of many built-in i18n features of Java and Flash. However, this approach offers the user a choice of language – an important option for science courses which are not always taught in the local language of the computer user.

Website Translation

As the international use of the simulations has grown to over 30 percent, there is an increased demand for local translations of the PhET website. The ERCSME at King Saud University required a fully translated mirror site. In addition, we found several independent groups who had translated the text in the PhET web pages and posted this translated version of the site to their own server. It is encouraging to see this level of interest; however these translated websites are static and cannot reflect updates to the simulations or other materials on the PhET site. Thus, together with

Fig. 3. The PhET website translation interface is shown. The left hand side lists each string in English and provides a text box for the translator to fill in. On the right, 2/3 of the screen shows the website with translated strings immediately visible. Blue boxes indicate translated strings, red strings that have not yet been translated and an orange box indicates a string that needs updating.

ERCSME, PhET undertook i18n of the PhET website. A cost-benefit analysis demonstrated that maintaining and adding necessary upgrades to the previous PHP–based site were nearly equal to the cost of creating an entirely new site using Java with the Apache Wicket framework (Apache Software Foundation 2010). Additional benefits would be improved speed and database handling.

Thus, a third complete rewrite of the PhET website began in August 2009 and is nearly complete. This new version went live in June 2010, and the translation feature will appear by fall 2010. Once implemented, this new version will have features including a very easy to use preview-able translation interface (Fig. 3), and it will allow the user to switch to and browse in their preferred language from any page.

One of the challenges was to allow translators to instantly preview their changes. A column of editable strings is displayed on the left side, while previews of the translated pages are displayed on the right. Whenever a string is translated or updated, the corresponding preview is updated, which quickly gives the translator feedback. Additionally, the website needs to handle regular modification or addition of content that does not happen in simulations. Any new strings are marked as "un-translated", and when an English string is changed, any translated versions are marked as "out-of-date". Email notification of these events to translators will allow them to keep their translations up-to-date.

Internationalization of Scientific Material

Arabic Translation Experiences

Here we discuss the work involved in translating PhET simulations from English into the Arabic language. Our experience again confirms that an expert in scientific materials with bilingual knowledge is needed in order to have accurate and understandable translations of scientific simulations. We faced several challenges in translating PhET simulations from English into Arabic that are shown to be common to other languages and could apply to any online scientific materials.

First, one of the biggest challenges is the selection of words to describe scientific quantities and expressions. These words need to be commonly used and understood by students and teachers across the Arabic-speaking countries. However, this has proven to be difficult. Words that are widely used in one country to express scientific quantities are not necessarily used by students and teachers in another country due to differences in scientific background and educational systems. For example, "acceleration" can be translated in two different ways. Some students are familiar with one, but not the other. One has to be careful when translating scientific materials to avoid possible misconceptions. For instance, a word might be correct linguistically, but might not represent an appropriate description of the intended scientific meaning. Words like "distance" and "displacement" used in mechanics should be carefully translated to reflect their scalar and vector meaning. This again shows why it is best to have a teacher do the translations.

To complicate things further, even in the same country, educational systems vary between high school and college level. For example, in Saudi Arabia, math equations and scientific notation are expressed using the Arabic language at the high school level but in the English language at the college level. To overcome this problem, a specific version for each Arabic-speaking country was created and labeled using a locale that includes the language and country code, and a more universal version for use in college is labeled with just the language code.

Moreover, different conventions for expressing abbreviations of units in Arabic exist, and for that reason, abbreviations were mostly avoided in the general versions of the simulations, unless they were explained, to make sure variables and units can be understood. One drawback to this solution is that when abbreviations are avoided, strings might be too long to fit in the allowed space in the simulations. Again, a possible solution is to create different versions for each locale that uses the specific abbreviations and units used in teaching science in classes.

Since the Arabic language is written from right-to-left, the direction of equations and scientific notation also have to be written right-to-left. Handling this was a challenge when using the Translation Utility as it involves shifting between English and Arabic to embed Arabic text in html codes and between brackets. The mixing of two languages with mixed directions in the same string makes it difficult to follow. In response to these difficulties, the Translation Utility includes the ability to individually preview strings with these HTML and MessageFormat syntax for debugging purposes. This individual string preview, along with the ability of running a translated version of the simulation from the Translation Utility, saves time and avoids possible layout problems (Fig. 2B). For equations, the reverse direction complicates writing equations from right-to-left if one equation consists of multiple strings as that might reverse order. A single string is preferable because it allows the translator to accommodate the right-to-left direction; however, sometimes a single string can become long and cause layout problems. These pros and cons must be considered with each individual case.

Chemical symbols (i.e. Na for Sodium) were kept in Roman letters as it is the standard in the Arab world. Greek symbols used in science and math, the most obvious being "π", are used in some cases and other abbreviations for symbols that are commonly used in Arabic textbooks were used in others.

For number formats, Arabic-speaking countries use two different systems: Arabic numerals (0, 1, 2, 3, …) used in the West, and so-called Hindi numerals or Eastern Arabic numerals (٠ , ١, ٢, ٣, …). Some Arab countries use only Arabic numerals while others typically use the two systems interchangeably. We decided to use Arabic numerals rather than translating to Hindi since nearly all students understand the Arabic numerals and additionally with Hindi, it's easy to confuse the zero (٠) and a decimal point (.). While text in Arabic runs from right-to-left, conversely numerals run from left-to-right.

Ongoing Challenges

Many of the challenges faced by the co-authors from Saudi Arabia are common to other languages as well. Languages that are written right-to-left clearly pose complications when translating from a language such as English that is written left-to-right. Different character sets bring with them all sorts of font problems. PhET initially incorrectly assumed that conventions for expressing equations, variables, and mathematical symbols were universal and later had to make these translatable. It was a pleasant surprise to find that chemical symbols (eg. Na for sodium) are universal. So far, all translators and educators, we have spoken with, have used the IUPAC (International Union of Pure and Applied Chemistry) system for name and symbol of chemical elements (IUPAC 2008).

As mentioned previously, there are some aspects of i18n that are not addressed by PhET simulations. Number formats, for example, are not handled properly. Some of these issues would be addressed by using Java's standard method of setting the locale for the Java Virtual Machine (JVM). If the JVM knows the locale, then things like number formats will be automatically handled by Java. Passing this information to the JVM requires the JNLP files request permission

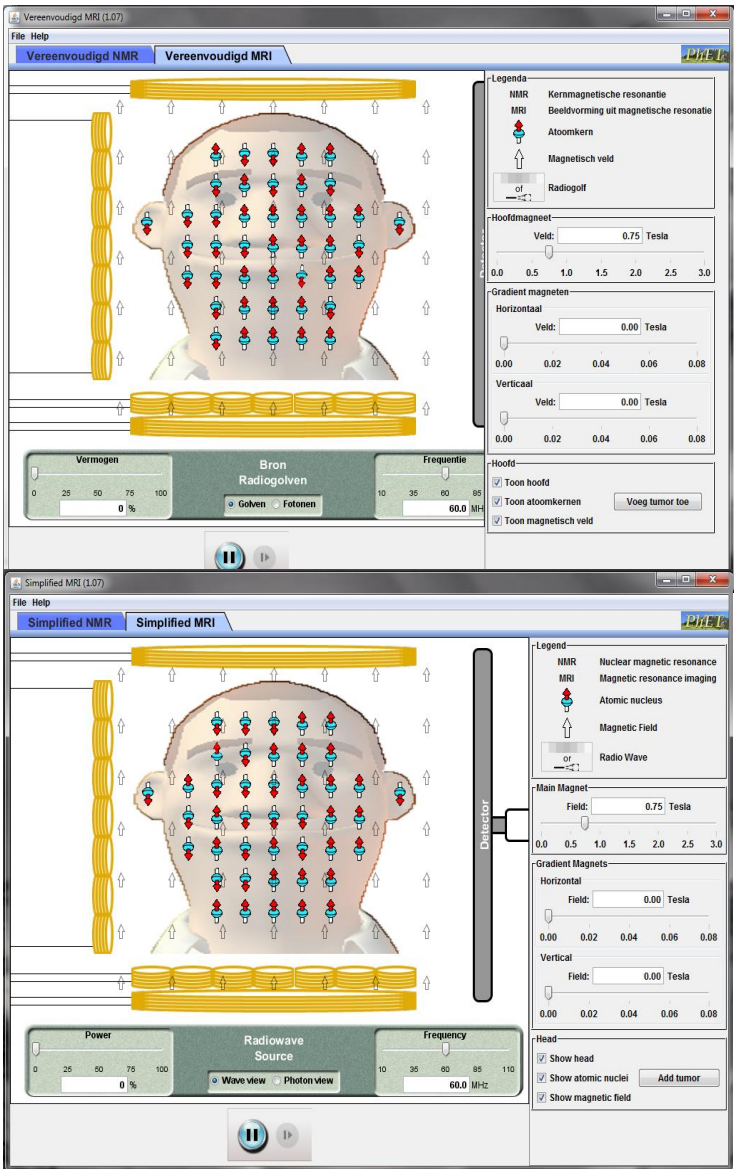

Fig. 4 shows the "Simplified MRI" Simulation translated into Dutch (upper). The translated strings on the lower center controls are too long and in the right hand control panel causing the control panel to cover the detector. The English version is shown in the lower screen shot demonstrating the intended layout.

to set Java System properties, and this in turn requires online users to accept a digital certificate. Several years ago we found that acceptance of digital certificates could pose a barrier to simulation use especially in schools where user privileges are extremely limited. As such, PhET has prioritized ease of online use and avoided solutions that require certificates. As certificates become more common, we may reevaluate this decision.

Translation of common strings has been another ongoing challenge. Strings that are common to many simulations are stored in special common-string files. These strings can be translated using the Translation Utility, but they cannot be tested because they have no associated simulation. This presents challenges for translators, since they have no context in which to evaluate translations of common strings. It also presents challenges for integrating and testing translations, since submitted translations for common strings must be integrated with all simulations. PhET is still refining the process of translating and integrating common strings.

Features that were originally intended for one simulation often are found to be generally useful, and are then migrated to PhET's common framework. If these features have associated strings, then the translations of those strings also need to be migrated from sim-specific string files to the common-string files. That migration is currently an expensive manual process. PhET is investigating how to automate this process

While the Translation Utility provides the ability to test translations, it does not detect user-interface layout problems. Layout problems can occur when a translated string is significantly longer or shorter than the original English string (Fig. 4). Longer strings in particular can cause part of the simulation's user-interface to overlap or become unusable. A translator who is not thoroughly familiar with a simulation may not notice these issues.

Conclusion

The PhET Project has created the Translation Utility for translating PhET Interactive Simulations from English to any other language and a system for delivering these translated simulations. The process now works quite well, although PhET discovered problems with many of their initial ideas. PhET learned many things about local conventions across the globe for expressing chemical symbols, scientific-notation, mathematical symbols, abbreviations and numerals, and the importance of teacher translators. Currently scientists/teachers have contributed a total of 1,758 translations in 51 different languages that are hosted on the PhET website. We hope that sharing this experience will benefit others interested in translating science education materials.

Acknowledgements This work would not have been possible without the expert simulations created by The PhET Team, particularly its superb software developers Sam Reid, Chris Malley, John Blanco, Mike Dubson, and Jonathan Olson. PhET is supported by the National Science Foundation, the William and Flora Hewlett Foundation, The Excellence Research Center of Science and Mathematics Education (ERCSME) at King Saud University, JILA and the University of Colorado. ERCSME is grateful for the Ministry of Higher Education and King Saud University, Riyadh, Saudi Arabia for their support.

References

Apache Software Foundation (2010). Apache Wicket http://wicket.apache.org

Deitsch, A. and Czarnecki, D. (2001). *Java Internationalization*. M. Loukides Ed. Sebastopol, CA: O'Reilly & Associates, Inc.

IUPAC (2008) http://old.iupac.org/index.html

International Organization for Standardization (2010). ISO 3166-1 Country Code decoding table http://www.iso.org/iso/iso-3166-1_decoding_table.html

International Organization for Standardization (2007). ISO 639-1:2002 http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22109

Oracle (2010). Java™ 2 Platform Standard Ed. 5.0 Class MessageFormat http://java.sun.com/j2se/1.5.0/docs/api/java/text/MessageFormat.html

PhET Interactive Simulations (2009). Research page: http://PhET.colorado.edu/research/index.php

PhET Team (2007). The PhET Translation Utility. http://phet.colorado.edu/contribute/translation-utility.php).

The Unicode Consortium. (2009). The Unicode Standard, Version 5.2.0, defined by: *The Unicode Standard, Version 5.2.* Mountain View, CA: The Unicode Consortium. (http://www.unicode.org/versions/Unicode5.2.0/)