

Chapter 5: Development of the Cognitive Items

Britta Upsing, Frank Goldhammer, Maya Schnitzler, Robert Baumann, Roland Johannes, Ingo Barkow and Heiko Rölke, DIPF; Thibaud Latour, Patrick Plichart, Raynald Jadoul and Christopher Henry, CRP; and Mike Wagner and Isabelle Jars, ETS

The implementation of the cognitive items for PIAAC faced several challenges. As PIAAC was the first international large-scale study to be conducted entirely on the computer, existing link items from prior studies IALS and ALL had to be converted from paper to computer. In addition, new items had to be developed both in literacy and numeracy to take advantage of the new possibilities of computer-based assessment. Further, an entirely new assessment domain, problem solving in technology-rich environments (PSTRE), was defined and items had to be developed. This was all done in a short timeframe in collaboration with participating countries that developed items on their own, as well as by combining item development teams from different countries.

To cope with these challenges, a multifaceted approach was taken, reusing existing item development and test delivery software to the extent possible and developing easy-to-use new software to fill in the gaps.

As a basis, the assessment software TAO was used (see Chapter 9). In the so-called electronic reading assessment (ERA) option of the PISA 2009 study, TAO was used with the Hypertext Builder, a graphical authoring tool for complex items. This approach was reused and extended for PIAAC, resulting in a completely new version called the CBA ItemBuilder. All in all, the following combination was used:

- Test definition, item sequencing, item questions: TAO
- Literacy linking items/stimuli: ItemBuilder
- New literacy items/stimuli: ItemBuilder
- Numeracy linking items/stimuli: ItemBuilder
- New numeracy items/stimuli: ItemBuilder
- PSTRE: TAO (new item type)

For PIAAC Round 2, a new requirement, to support the right-to-left languages Arabic and Hebrew, added new challenges to the development and delivery of the cognitive items. Whereas in Round 1 a combination of the CBA ItemBuilder software (for literacy and numeracy) and the

TAO/BLACK framework (PSTRE) was used to implement the CBA unit, in Round 2 a framework based on HTML5 was used.

The software and the procedures to produce the items are described in more detail below.

5.1 Development of literacy and numeracy items

As outlined above, literacy and numeracy items were produced using the CBA ItemBuilder software. For the linking items, the existing paper items were used as a draft, while the new items were built from scratch or drafted using other standard software.

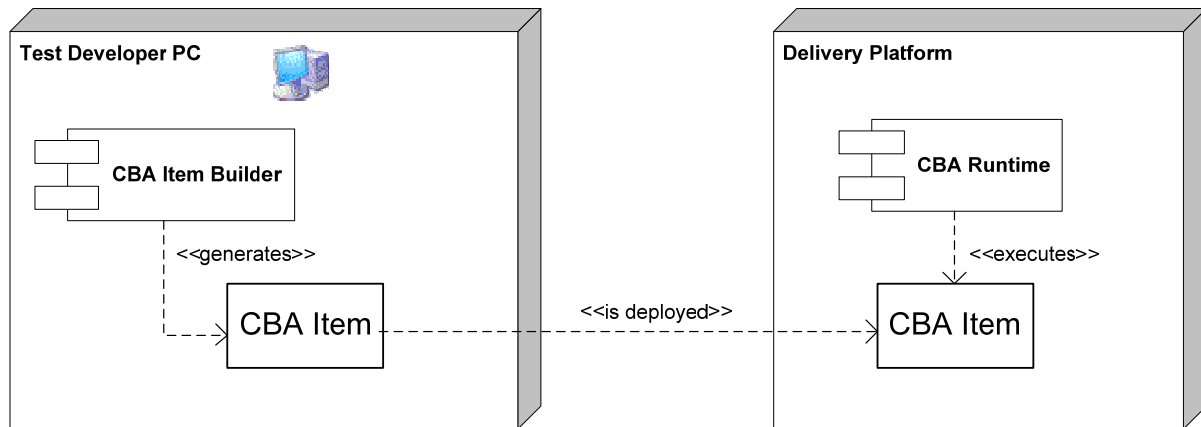
The goal was to produce an international *universal* item that could serve as a basis for country translations and adaptations without having to touch the layout. As it later turned out, this often was not possible as there are great differences in how much space different languages consume.

The CBA ItemBuilder is a graphical tool that enabled assessment domain experts to develop complex items in a what-you-see-is-what-you-get (WYSIWYG) manner without any programming. The CBA ItemBuilder consists of two major parts: the editor and an independent runtime environment. The editor allows for a WYSIWYG work style where you can drag item elements from a palette and freely drop them wherever necessary. Items designed in the editor are stored in an intermediate item description format. From this format, the executable item is generated using software generation principles. This ensures that the runtime for CBA ItemBuilder items can be changed without touching the editor in a relatively easy manner. The first versions of the runtime environment (used for PISA 2009 ERA) were based on Adobe Flash. This was changed in the preparation phase of the PIAAC study to a mix of standard Web technologies: HTML, JavaScript and SVG. The items produced with the CBA ItemBuilder could be used standalone as well as integrated into other assessment software. For simple assessment purposes, a graphical user interface was provided, allowing for editing and executing tests. In PIAAC, ItemBuilder items were used as stimuli integrated to TAO items. More information about the ItemBuilder software can be found in Rölke (2012). Here we only outline its possibilities.

5.1.1 The CBA ItemBuilder

The CBA ItemBuilder System consists of several interconnected components – see Figure 5.1 for an overview. The most prominent is the standalone CBA ItemBuilder. It is a desktop application based on Java and Eclipse technology. The CBA ItemBuilder was used to design and try out single items. Once an item was ready, a device-independent format was generated for later usage: the CBA Item. A CBA Item could be deployed on various platforms ranging from USB drives to the Internet.

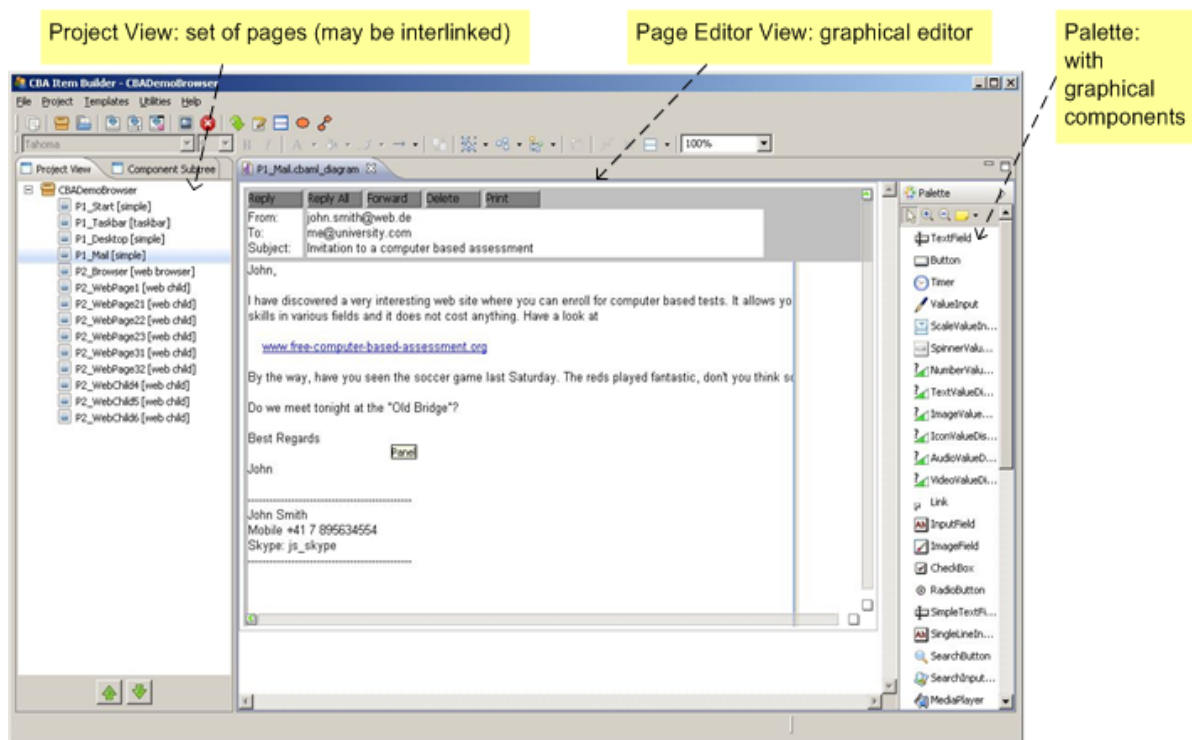
Figure 5.1: System architecture overview (M. Dorochevsky, Softcon)



CBA Items were delivered by a Java application server, usually JBoss. They could run on other servlet containers as well. As stated above, they consist of open Web standard: HTML code enriched by JavaScript and SVG. The Eclipse RAP framework (Eclipse Foundation 2012b) is used for graphical user interface components. Complex computations are done on the server side, implemented in Java. CBA Items can be displayed on any current browser (e.g., Firefox, Chrome, Internet Explorer), provided JavaScript is enabled. For PIAAC, only Firefox was supported.

The CBA ItemBuilder offers a graphical user interface to compose stimuli and items via drag and drop. A stimulus could be used in one or several items, for example, in combination with different questions. As with other modern integrated development environments, it offered different *views* on the item at hand. Figure 5.2 gives an example for a mail client item in process.

Figure 5.2: CBA ItemBuilder graphical user interface



On the left side of the CBA ItemBuilder window, the *Project View* is shown. This view gives an overview on complex items that consist of more than one page. The Project View allows for a quick selection of all *Pages* that belong to one item (or project). In the middle of the screen, the *Page Editor View* is shown. This is the most important view. It allows for authoring of item pages or stimuli – to put it simply, something that can be shown on a computer screen later on in an assessment. The screenshot in Figure 5.2 shows the item as it is being edited. This is done in a mode that comes close to WYSIWYG but abstracts from images or design effects, for example. These can be included in a preview of the item.

On the right side of the image in Figure 5.2 is the *Palette View*. It contains all elements an item can be constructed with, for example, text fields, buttons, or input fields. To apply such an element, the user simply drags it with the mouse and drops it in the Editor View. An example of an editing process can be found in the following subsection on so-called link items.

For an in-depth look at the CBA ItemBuilder, please refer to the mentioned literature (Rölke, 2012).

5.1.2 Link items

As the domains of literacy and numeracy had been measured in previous large-scale international surveys, it was a requirement that PIAAC link back to IALS and ALL. As a result, a set of linking items that had already been used in these studies was used in PIAAC. These studies have been performed as paper-and-pencil studies. The materials were therefore available as text documents in Microsoft Word format. The main challenge for the item development was to reproduce the

paper item layout as closely as possible. Naturally, this could not always be achieved. To give just one example of the problems, the layout had to be changed from portrait-format paper to a landscape-format screen, and scrolling was not allowed.

Some materials like texts and images could be extracted from the existing paper items. Some images had to be redone, to add interactive areas, for example.

Figure 5.3 shows an example of a typical link item. It was originally derived from a newspaper article and was formatted for the paper assessment. For PIAAC it had to be reformatted to fit the screen layout and so on. To come as close as possible to the paper item that required marking as a means for answering the question, a new interaction mode called *multiple highlighting* was introduced. We come back to this interaction mode later on when dealing with the scoring of the items.

Figure 5.3: Link item example

The screenshot displays a PIAAC assessment interface. At the top left is the OECD PIAAC logo. The interface is divided into two main sections. The left section, with a light blue background, contains instructions: 'Look at the list of preschool rules. Highlight information in the list to answer the question below.' Below this is a question box: 'What are the two rules about taking medicine to the preschool?'. The right section, with a white background, is titled 'Preschool Rules' and contains a welcome message followed by a bulleted list of rules. The rules are: 'Please have your child here by 9:00 am.', 'Bring a small blanket or pillow and/or a small soft toy for naptime.', 'Dress your child comfortably and bring a change of clothing.', 'Please no jewelry or candy. If your child has a birthday please talk to your child's teacher about a special snack for the children.', 'Please bring your child fully dressed, no pajamas.', 'Please sign in with your full signature. This is a licensing regulation. Thank you.', 'Breakfast will be served until 7:30 am.', 'Medications have to be in original, labeled containers and must be signed into the medication sheet located in each classroom.', and 'If you have any questions, please talk to your classroom teacher or to Ms. Marlene or Ms. Tree.'

OECD PIAAC

Look at the list of preschool rules. Highlight information in the list to answer the question below.

What are the two rules about taking medicine to the preschool?

Preschool Rules

Welcome to our Preschool! We are looking forward to a great year of fun, learning and getting to know each other. Please take a moment to review our preschool rules.

- Please have your child here by 9:00 am.
- Bring a small blanket or pillow and/or a small soft toy for naptime.
- Dress your child comfortably and bring a change of clothing.
- Please no jewelry or candy. If your child has a birthday please talk to your child's teacher about a special snack for the children.
- Please bring your child fully dressed, no pajamas.
- Please sign in with your full signature. This is a licensing regulation. Thank you.
- Breakfast will be served until 7:30 am.
- Medications have to be in original, labeled containers and must be signed into the medication sheet located in each classroom.
- If you have any questions, please talk to your classroom teacher or to Ms. Marlene or Ms. Tree.

5.1.3 New items

New items for literacy were developed by ETS in the U.S. in cooperation with ACER in Australia. These items used features of the CBA ItemBuilder that were not utilized in the linking items. The new items in literacy focused on electronic texts, including Web pages, emails and discussion

boards. This required the use of a simulated Web browser environment by the user. This environment was modeled in the CBA ItemBuilder. The ItemBuilder supported multiple stimulus “pages” within a single item. These pages could be linked via hyperlinks that were embedded in the item text. The runtime for these items supported maintenance of a hyperlink history, allowing the user to navigate back and forth among pages that had been visited.

The new items in literacy featured one unique response mode. Some items asked the user to click on a link in the text as his or her answer to the question. In these items, the scoring was based on the target of the hyperlink. Two pages were constructed, one for correct links and one for incorrect links. These pages looked identical to the test taker but had different identifiers internally. The test taker was given credit for a correct answer if he or she finished the item with the correct page showing in the browser.

New items for numeracy were developed by ETS in cooperation with members of the Numeracy Expert Group. These items had similar functionality and scoring mechanisms as the linking numeracy items. The only thing that distinguished these items from the linking items was the use of color images and artwork. The linking items, because of their legacy as paper-based items, were entirely in black and white.

5.2 Development of the automatic scoring software

To enable adaptive testing in PIAAC for literacy and numeracy, those items had to be scored automatically and instantaneously by the platform. Various response modes were used that required developing different strategies for automatic scoring of test-taker responses. If a response mode included scoring rules with any textual or numerical information, they had to be adapted nationally. For those response formats, various workflows and (online) tools were developed to organize and support the national adaptations and the testing of adapted scoring rules.

5.2.1 Response modes in literacy and numeracy

Out of the PIAAC domains, literacy and numeracy included automatic scoring for adaptive testing and were based on a variety of response modes. Response modes could be divided into: i) those requiring interactions with the stimulus; and ii) those including interactions with the left panel of the PIAAC screen as shown in Figure 5.4. The stimulus interactions were:

- Stimulus highlighting (items requiring the test taker to select a piece of text by clicking and dragging or by double-clicking a word)
- Stimulus clicking (meaning it is necessary to click on a graphical element, which usually becomes marked to indicate it has been selected; sometimes more than one element is clickable)
- Stimulus clicking link (used for new literacy items and refers to clicking on a link in a simulated Web browser environment)
- Stimulus multiple-choice check box (clicking one or several check boxes that are included in a simulated Web browser environment used in new literacy items)

Figure 5.4: Scoring a highlighting item

The screenshot shows the PIAAC (Programme for International Adult Assessment) interface. The top header is blue with the OECD PIAAC logo. The interface is split into two main panels. The left panel is light blue and contains instructions: 'Look at the list of preschool rules. Highlight information in the list to answer the question below.' Below this is a question box: 'What are the two rules about taking medicine to the preschool?'. The right panel is white and titled 'Preschool Rules'. It contains a welcome message and a list of rules. Some text in the rules is highlighted in yellow, including 'original, labeled containers', 'signed into the medication sheet', and '7:30 am'.

OECD PIAAC

Look at the list of preschool rules. Highlight information in the list to answer the question below.

What are the two rules about taking medicine to the preschool?

Preschool Rules

Welcome to our Preschool! We are looking forward to a great year of fun, learning and getting to know each other. Please take a moment to review our preschool rules.

- Please have your child here by 9:00 am.
- Bring a small blanket or pillow and/or a small soft toy for naptime.
- Dress your child comfortably and bring a change of clothing.
- Please no jewelry or candy. If your child has a birthday please talk to your child's teacher about a special snack for the children.
- Please bring your child fully dressed, no pajamas.
- Please sign in with your full signature. This is a licensing regulation. Thank you.
- Breakfast will be served until 7:30 am.
- Medications have to be in original, labeled containers and must be signed into the medication sheet located in each classroom.
- If you have any questions, please talk to your classroom teacher or to Ms. Marlene or Ms. Tree.

The interactions on the left panel of the PIAAC screen were:

- Left-panel single-choice radio button/pulldown menu (used in new numeracy items and refers to clicking and selecting a single item in a group of radio buttons or in a pulldown menu, respectively)
- Left-panel multiple-choice check box (clicking one or several check boxes provided on the left panel)
- Left-panel numeric entry number match/exact match (required the test taker to enter number(s) into input box(es))

National adaptations were done mainly for: i) highlighting items because of the translation of textual information; and ii) numeric entry items because of adaptations to the national number format and/or currency system. Therefore, scoring testing efforts were focused on these item types. In contrast, clicking, multiple-choice and single-choice items were translated and adapted without affecting the scoring definition, that is, it was the same as in the international version. In these cases, errors were assumed to be less probable because adaptation of scoring rule as one source of error was not relevant.

5.2.2 Automatic scoring of highlighting items

Stimulus highlighting items required the test taker to select one or more pieces of text by clicking and dragging or by double-clicking a word.

Highlighting items were usually scored by evaluating whether the minimum correct response had been selected, and, at the same time, whether the selection did not exceed the maximum correct response. Hence, each highlight item had a specified minimum and maximum response. The minimum response consisted of individual words or phrases that were identified as critical portions of a correct response. In general, the maximum correct response for highlight items included the entire line in which the correct answer was located, any or all of the line above the correct response, and any or all of the line below the correct response. However, the maximum correct response could not contain any incorrect information, as identified by test developers. So if any part of the line above or below had contradictory or incorrect information, it was excluded from the maximum correct response.

Highlighting responses were scored automatically by the system based on the definition of text blocks and the scoring rule referring to the text blocks. Text blocks defined the parts of the text in the stimulus representing the minimum and maximum correct response. They were not visible for the test taker. Text blocks with the correct answer were already defined in the international item version. For the national versions of the stimulus, the text was translated and the position and size of text blocks were adapted by using a specific text block editor built into the CBA ItemBuilder software.

5.2.3 Automatic scoring of numeric entry items

Exact match items required the test taker to enter number(s) into input box(es) on the left panel of the PIAAC screen. Numeric entries were scored automatically by the system based on the definition of correct numeric response(s) included in the scoring rule. The exact match scoring method was equivalent to string match; that is, the system checked for character equivalence instead of numerical equivalence. For example, if a correct response for an exact match item was defined as “5”, an entry of “20/4” would also have been scored as incorrect.

Number match items also required the test taker to enter number(s) into input box(es) on the left panel of the PIAAC screen. Numeric entries were scored automatically by the system based on the definition of correct numeric response(s) included in the scoring rule. The scoring method “number match” means that the response is correct as long as it represents the correct numerical value, regardless of the way the number is “spelled” by the test taker. For example, if a correct response for an exact match item was defined as “5”, an entry of “20/4” would also have been scored as correct.

For some items, to retain realism, the magnitude of numbers and/or the number format were adapted for the national version. In this case, scoring rules also were adapted.

The Round 1 Field Test scoring approach for number match items was revised to address several concerns expressed by experts and countries. In particular, the handling of decimal separators in the Field Test was considered to be too strict and unrealistic. Thus, the Main Study scoring approach introduced “double scoring,” which means that – within the system – the test taker’s response to an item was scored twice. So, before the system gave a final evaluation of the test

taker's answer, in all country versions it went through the following two scoring steps: i) The first scoring assumed a comma as decimal separator (i.e., acceptable thousands separators were blanks or periods), while ii) the second scoring assumed a period as decimal separator (i.e., acceptable thousands separators were blanks or commas). If at least one of the two scorings yielded a "correct," the response was considered correct.

Moreover, for the Main Study scoring, a so-called "strong mode" of the thousands separator(s) check was activated. This means that if the test taker used a thousand's separator, the position of the separator needed to be correct to be considered a correct response. In general, only groupings by three digits were accepted. Groupings by four digits were acceptable only in the Japanese version (but not mixed groupings by four and three digits within a single number).

5.3 Scoring testing strategy

The automatic scoring procedures of the international versions of literacy and numeracy units were tested by the Consortium prior to distribution of the national versions. National language versions needed to be tested again by countries thoroughly, because for many items the definition of correct response(s) was adapted. This meant that scoring testing at the national level was especially important when the correct response included translated and/or adapted textual and numerical information.

The general rationale and procedure established for testing the international version were also the basis for testing the national version. They were revised iteratively during the international testing process. Basically, two sources of error were observed during international testing: i) errors at the level of item editing, that is, the scoring information was specified incorrectly by the item editor (specification error); and, ii) errors at the level of technology, that is, the software did not work accurately (implementation error). All detected errors were fixed, and the scoring of affected units was tested again successfully.

The testing was done manually, meaning the person responsible for testing completed a unit and item, respectively, as the test taker was supposed to do. Depending on the response mode, the tester used the keyboard for numeric entries or the mouse for selections to complete each item. For each response mode, a set of testing steps including the expected scoring result was defined to cover the most important test cases. When an item was tested, the tester gave a response as required by the testing step and compared the observed scoring result and the expected scoring result. Discrepancies between the observed scoring result and the expected scoring result needed to be documented and reported to the Consortium for debugging and for the Consortium to provide a revised version in the following testing iteration.

Countries were required to follow the international testing plan and for customizing the international test cases to their national versions as explained by the Consortium.

For the Main Study version of numeric entry items, implementing the adaptations and testing the adapted scoring rules was done centrally by the Consortium.

5.4 PIAAC Round 2 cognitive Items

For PIAAC Round 2, a new requirement, to support the right-to-left languages Arabic and Hebrew, added new challenges to the development and delivery of the cognitive items. It was decided to

re-implement all of the cognitive items, including those for literacy, numeracy and PSTRE, using HTML5 based technologies. As the PIAAC tests were already delivered using a web browser, and HTML already had wide support for right-to-left languages, reimplementing items was determined to be a better long-term solution than updating the existing technologies. Additionally, the HTML5 framework to be used, called “NAX,” had already been utilized successfully for the Programme for International Student Assessment 2012, which had many of the same requirements as PIAAC for rendering and scoring of CBA tests in multiple languages.

The first step was to reauthor all English master units using the NAX technology. This was done using standard HTML markup and CSS stylesheets by Web programmers. While it was not feasible to create identical copies of the Round 1 CBA units, every effort was made to come as close as possible. For instance, it was not possible to maintain exact matches for line breaks in texts, but general length of lines and width of margins were maintained.

While the visual representation of the PIAAC units was being implemented, other Web programmers needed to develop code libraries to support the interactivity needed for the PIAAC tests. Dynamic functionality within the units, such as highlighting in literacy and the simulated tools in PSTRE, was implemented using JavaScript, which executed within the web browser. Data management (saving response data) and navigation through the units of the test were implemented using a combination of JavaScript and PHP.

5.4.1 Highlighting responses

An important response mode for PIAAC is highlighting within the literacy units. For PIAAC Round 2, the basic functionality of text highlighting from Round 1 was replicated. In particular, the ability to highlight multiple, separate regions of text was possible. If the selected texts overlapped, the highlighted regions were merged into one region. Clicking once on a highlighted region unhighlighted/deselected a selected text.

For Round 2, an enhancement was made to the text highlighting mechanism. In Round 1, a user could select individual characters in the text. Thus parts of words could be selected. This led to errors in scoring because correct answers were defined in terms of complete words. Therefore, in Round 2, the highlighting mechanism was improved so that only full words could be selected. A respondent would use the mouse to select text as usual, selecting at the character level. Once the mouse button was released, indicating the completion of the highlighting operation, the beginning and ending of the highlighted region were expanded to the next word boundary.

For example, if the user released the mouse button after highlighting the following text:

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.

the system would automatically expand the highlighted region as shown below:

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.

5.4.2 Automatic scoring

Due to the change in technologies for rendering cognitive items in Round 2, the automatic scoring logic also needed to be rewritten. This code was written in 100% JavaScript and ran within the Web browser. A custom scoring library was created for each response mode used in PIAAC:

- Stimulus highlighting
- Stimulus clicking
- Stimulus clicking link
- Stimulus or question multiple-choice check box, radio button, dropdown menu or numeric response
- Custom JavaScript (for each PSTRE unit)

Scoring for each item was specified using rules saved in an XML file and expressed in a programming as syntax. For example, to score a multiple-choice question, a rule such as the following would be used:

```
<scoring source="question" mode="form" item="item1">
  <![CDATA[ { INTERVAL(PARSEFRAC(C910AC01_NUM1),[77.4,77.6]) } ]]>
</scoring>
```

This rule indicates that the value tagged as “C910AC01_NUM1” should be parsed as a number (this supports both fraction and decimal formatting) and then checked to see if it is between 77.4 and 77.6.

In such cases where the response is numeric and the response was interpreted as a number, the software would try multiple ways to interpret the response. It would use different decimal separators (period and comma) and thousands separators (decimal, comma, space) to see if any of the interpretations led to a correct response. If so, credit was given for a correct response, and if all attempts failed, the response was reported as incorrect.

The logic for scoring the multi-highlight response was similar in Round 2 to Round 1. For each item, text blocks were defined to indicate the text needed for the minimum correct response. Other text blocks were defined to indicate text that could not be part of a correct response. The scoring rule was then defined so that some set of text blocks had to be completely selected (the minimum correct response) and another set of text blocks could NOT be partially selected.

The text blocks used for the multi-highlight scoring were defined on a per language basis. A tool was developed that allowed countries to adapt the text blocks from the master version of a unit to accommodate local translation. After adapting the blocks, the tool allowed users to test the text-block definitions by highlighting text and checking to see if it was scored as correct or incorrect. Verifiers and other contractors used this same tool to check the work of countries as part of quality control procedures. Additionally, countries and contractors could use this tool to check adaptations (made centrally) to scoring rules in numeracy that needed to be adapted due to differences in currency values.

References

Rölke, H. (2012). *The ItemBuilder: A graphical authoring system for complex item development. In World conference on E-Learning in corporate, government, healthcare, and higher education (ELEARN)* (pp. 344–353). Waynesville, NC: Association for the Advancement of Computers in Education.

Chapter 6: Development of Technical Support Tools

Britta Upsing, Frank Goldhammer, Maya Schnitzler, Robert Baumann, Roland Johannes, Ingo Barkow and Heiko Rölke, DIPF; Thibaud Latour, Patrick Plichart, Raynald Jadoul and Christopher Henry, CRP; and Mike Wagner and Isabelle Jars, ETS

6.1 Development of the Item Management Portal

The Item Management Portal was the central PIAAC portal for all aspects of item development, item management, translation, adaptation, scoring and layout testing, and so on. It consisted of several parts interconnected by several workflows. Workflows were available only to authorized roles (or users) of the Item Management Portal. Therefore, a user and rights management system was also available. (For example, a translation of an item could only be verified by an authorized verifier, and only after it had been released by the respective NPM.)

The Item Management Portal offered a multitude of different views to the central repository of all item-related information in PIAAC. It encapsulated a central database, a file server, a TAO Installation, and a CBA ItemBuilder server installation, and it managed hundreds of gigabytes of data during the PIAAC study.

We cannot give a full overview of all aspects of the Item Management Portal here. Therefore we concentrate on specific parts to give an idea of the overall functionality.

Figure 6.1 shows the so-called Welcome Screen of the Item Management Portal that was displayed after a NPM logged into the portal.

As the Item Management Portal was about items, several possibilities were on hand to access single items or for an overview of several items. For example, in the *Groups* box on the left, it was possible to select all link items in the field of literacy with one click. From those you could, for example, select a single item in the *Units* box on the right.

Figure 6.1: Item Management Portal *Welcome Screen* for NPMs

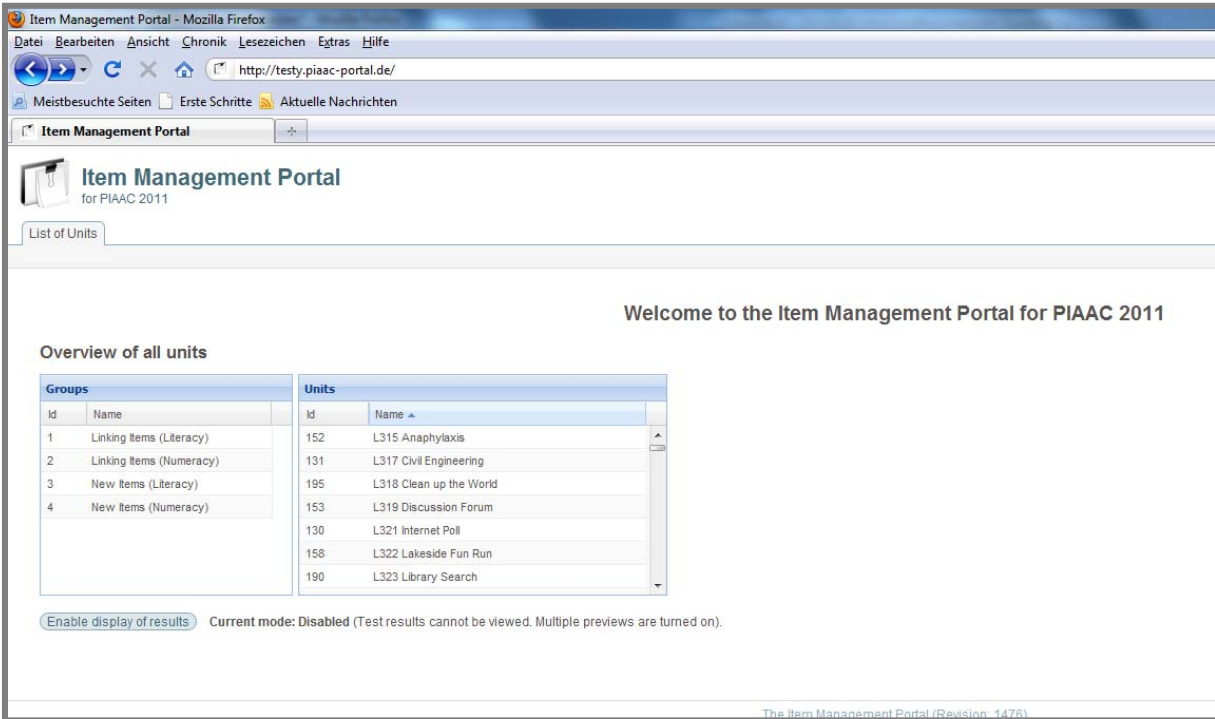


Figure 6.2: Item Management Portal with item selected

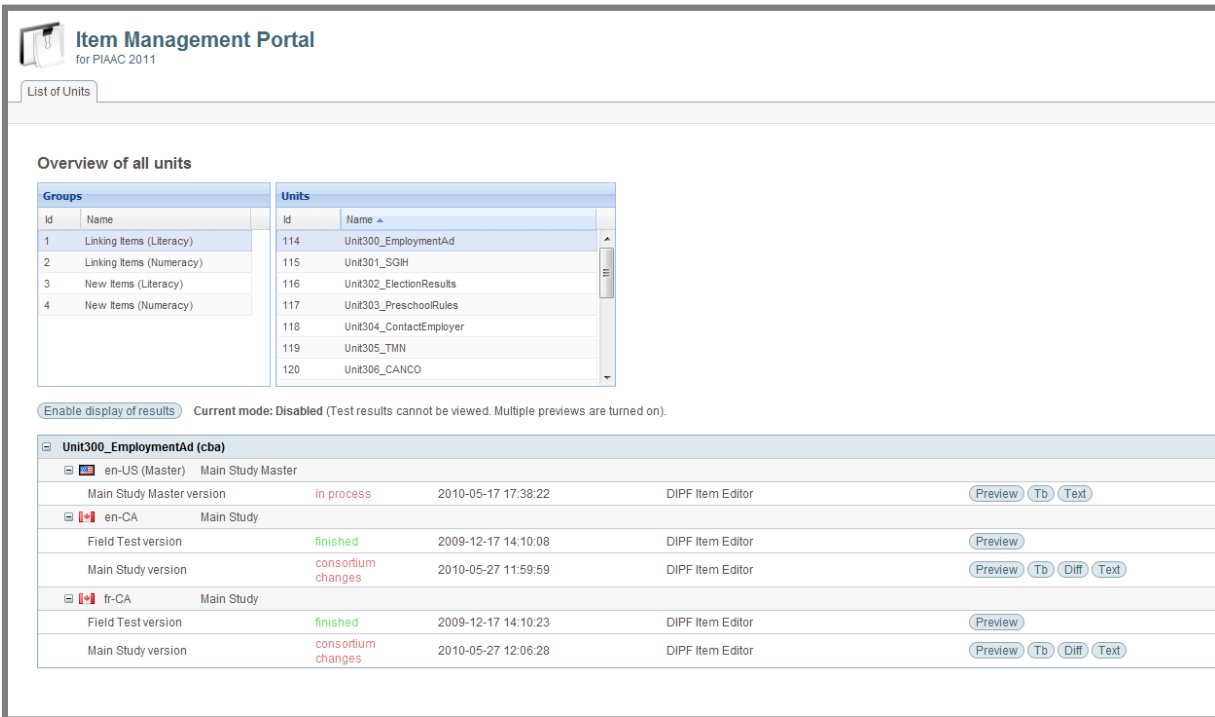
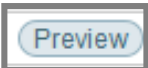



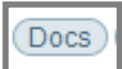
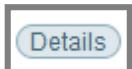
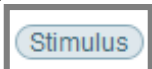


Figure 6.2 shows the Item Management Portal still in the NPM view, with one item selected. For this item, information about status and availability were displayed and several options were on

hand depending on the step and the units' status in the overall process. The following list, for instance, shows the options that were relevant for preparing the Main Study version by revising the Field Test version:

- If you clicked on the  button, you would see the original version of the stimulus.
- If you clicked on the  Button, you would see a list of the text blocks included in the stimulus.
- If you clicked on the  Button, you would see all the changes made in the stimulus.
- If you clicked on the  Button, you would see a clean text version of the stimulus.

There were also several more buttons not seen in Figure 6.2.

- If you clicked on the  Button, you would see the discussion between your country and the Consortium.
- If you clicked on the  Button, you would see the different inquiries that the stimulus consists of; you could choose the one you needed to edit it.
- If you clicked on the  Button, you could download the stimulus.

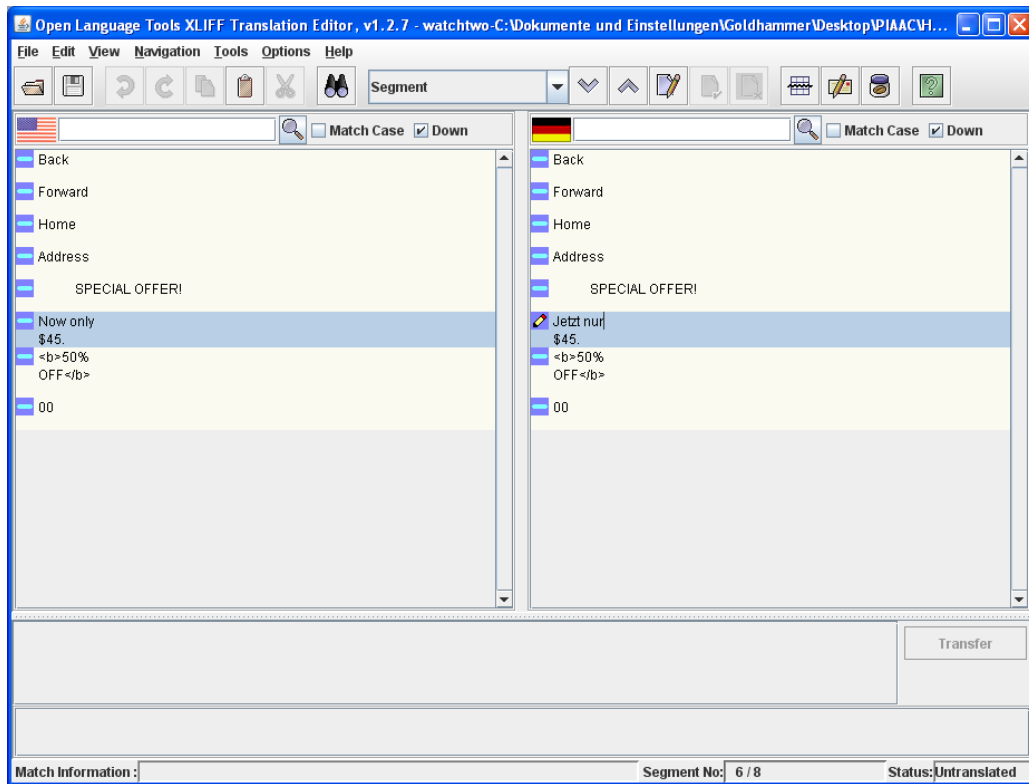
Note that this is just a small part of the functionality of the Item Management Portal. Depending on your role and the status of the item development, other options were available such as upload and download of translation files, commenting, releasing items to the next step of the workflow and many more.

6.2 Translation and adaptation tools

For translation and adaptation of items, external tools were used rather than the Item Management Portal itself. This was to support offline work of these sometimes lengthy and time-consuming tasks without the need of a steady and reliable Internet connection. Two tools were provided: the Open Language Tool (OLT) and the Textblock Translation Editor (TBTE).

The OLT, shown in Figure 6.3, is an open-source translation editor implemented in Java originating from the company *Sun*. It had already been used for the Programme for International Student Assessment 2009 study and has been further developed since. The OLT builds upon the XML Language Interchange File Format (XLIFF), an XML standard for translations. The CBA ItemBuilder provides a built-in support for XLIFF. Doing the translation externally to the item authoring software rather than in the CBA ItemBuilder itself offered the advantage of limiting layout changes by translators to an absolute minimum. Further information about OLT can be found at <http://open-language-tools.java.net/>.

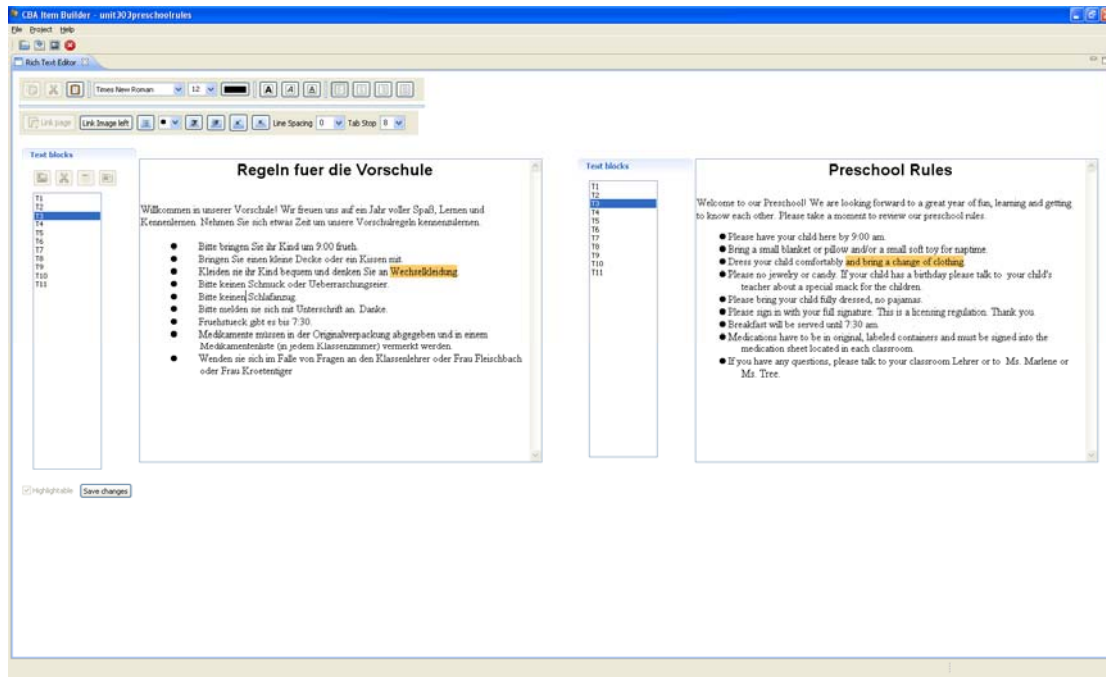
Figure 6.3: Open Language Tool



The purpose of the TBTE was to set the scoring definitions in translated highlighting items in accordance with the scoring definitions in the international item versions. The tool was to be used by the National Reviewer (NPM) in the context of setting the scoring definitions of a unit. In the workflow of the Item Management Portal, this step took place between reconciliation and verification. The TBTE essentially is a restricted version of the CBA ItemBuilder that does not allow for any change of layout other than changing text blocks. This is illustrated in Figure 6.4.

The TBTE was used to transfer the text blocks defining correct and incorrect parts of a highlighting interaction – see the chapter on scoring for details. The user could see both the original text and the translation and select one text block after the other. On the right side, the original text block definition was shown. This was to be transferred to the translated text on the left side of the screen. Afterward the new text blocks became an integral part of the automatic scoring of the translated item.

Figure 6.4: Textblock Translation Editor

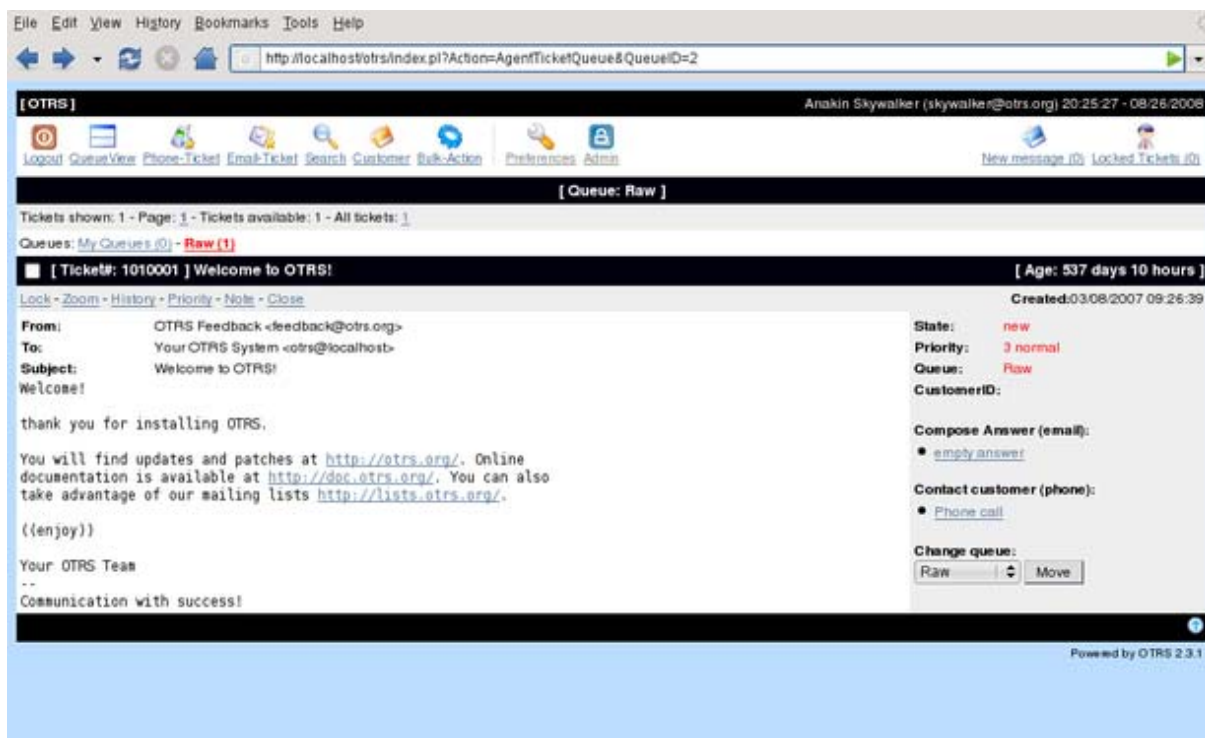


6.3 Support tools

Support for users and stakeholders in PIAAC was mainly provided online, via mail and preferably through the ticketing portal OTRS (Open Ticket Request System). OTRS is an open source problem reporting system. More information is available at <http://www.otrs.com>.

Figure 6.5 shows an example view of the OTRS system. The image is taken from the OTRS user manual. The OTRS system could be used via any browser and/or via mail exchange. During the field phases, additional phone support was offered. The phone support was backed by the OTRS to make sure all support cases were stored in a single location for later reference.

Figure 6.5: Generic OTRS ticket (from OTRS Manual)



6.4 PIAAC Round 2 updates

In PIAAC Round 2, some changes were made to the set of technical support tools, primarily due to the change in the technology used for implementation of the CBA units. One change was to replace the Item Management Portal with a new PIAAC Translation Portal. This portal was used for managing the translation process for all materials, including CBA units, PBA booklets, BQ sections and the interview workflow. The portal enforced a series of steps in the translation process (such as translation, reconciliation, verification, etc.), and each step was assigned to specific user accounts with specific roles (e.g., translator, reconciler, verifier). These translation processes were implemented using the TAO workflow engine (see section 9.2.2 for more information).

Users of the portal could download and upload files needed for translation as well as preview CBA units, both in their source and translated forms. These previews were live versions of the units and were generated on demand when a new copy of a translated XLIFF file was uploaded for a unit.

The Translation Portal also implemented a new Scoring Review capability, allowing users to test the scoring of items in the CBA units. This was important for items where scoring needed to be adapted on a per language basis. Integrated into this Scoring Review capability was a tool for adjusting the text blocks used for scoring highlighting items in literacy. This tool, similar to the TBTE of Round 1, was completely reimplemented due to the different technology used for the CBA units.

The other technical support tools (OLT, OTRS) were not changed for Round 2.

Chapter 7: Development of the CAPI Questionnaire System

Thibaud Latour and Raynald Jadoul, CRP; and Mike Wagner, ETS

7.1 Introduction

This chapter describes the CAPI system that was used to implement the BQ of the PIAAC survey from a design and operational viewpoint. Because the CAPI system was built on TAO, and because TAO is intrinsically built around knowledge technologies, the general rationale of the BQ was for it to operate as a knowledge elicitation tool organized as an interview and implemented as a workflow (WF).

In knowledge technology terms, elicitation consists of populating a general model with everything one wants to know about particular topics, with specific data on particular observations. Technically, the information sought can be formalized as a knowledge model called an ontology. In a nutshell, an ontology formalizes a shared understanding of a domain by identifying a hierarchically ordered set of abstract concepts, the relationships tying those concepts together, and the properties that characterize them. In addition, concrete instances of the model are considered to describe particular individuals defined by the abstract part of the ontology. Giving a concrete example of a concept by creating a corresponding individual defines its nature. Specific values for each of the properties are assigned for concepts. The descriptions of these individuals are also part of the ontology.

In PIAAC, the CAPI system was conceived as a tool to describe these individuals as instances in an ontology corresponding to the BQ framework. Building such a tool involved creating the ontology with the knowledge domain corresponding to the BQ framework and properties of concepts associated to BQ variables; associating to properties some tools to collect the instantiation values (the questions and the questionnaire); organizing these tools in a consistent sequence (the interview process); and reporting the collected data. The following sections describe in details the specifications of the CAPI system to deliver the BQ in the household and how it was implemented into TAO.

7.2 CAPI system specifications and features

7.2.1 Overall description of the system

The overall goal of the BQ CAPI system was to support the data collection in households during the survey as well as support the creation of complex questionnaires. The system was designed for a series of different users such as system administrator, BQ authors and interviewers. To specify such a tool, and in order to capitalize on the experience gained in the ALL survey, a group was set up by the PIAAC Consortium gathering specialists from the Australian Bureau of Statistics,

Statistics Canada and Westat working in close collaboration with the teams at the Research Institute Henri Tudor, DIPF and ETS.

From a functional point of view, the CAPI system included a series of components supporting the authoring of the different elements of the questionnaire, that is, the questions and possible answers, as well as the specification of the question sequencing, along with translation in all the PIAAC languages and maintenance of successive versions. The system also supported the interview by presenting the questions to be read to the respondent by the interviewer according to the sequence defined by the BQ designers. Because countries had different constraints and processes regarding their national surveys and privacy regulations, the CAPI system enabled specific initialization decided at the level of the National Centers (NCs). The system also proposed a series of tools to help the interviewer and collect his or her remarks during the interview process.

The whole system supporting the PIAAC data collection on the field was run on laptops. As for other technological components, a series of technical and operational requirements were specified both for the implementation of the CAPI system and for the countries that had the responsibility to buy, set up, operate and maintain the hardware devices for their interviewers. Among these elements, security issues in terms of data confidentiality and integrity were scrutinized carefully. The usability, including the careful internationalization of the system for the interviewer, was also considered as they constitute an important factor that impacts the quality of the collected data.

Some countries were not equipped with a case management system suitable for PIAAC. To provide basic case management capabilities to these countries, basic functionalities were also envisioned.

The CAPI system was a standalone system that ran on a virtual machine (VM) within a laptop. To ensure that the countries were equipped with hardware that was sufficiently powerful with respect to the system, a questionnaire was elaborated for countries to describe the hardware they expected to use. In general, the system was not too demanding and modern laptops were sufficient, with a preference for fast CPU machine to accelerate the latency of the graphical user interface. Indeed, in order to control the overall duration of the interview, to keep the respondent focused, and to enable the interviewer to chain questions fluently, the Consortium fixed the maximum acceptable duration of the period between the validation of an entry and the display of the following question at two seconds. This latency included all intermediate response processing, that is, for branching or preparing precomputed responses.

The Linux Debian Open-Source Operating System (OS) was chosen for the VM and VMware, more particularly VMware Server – compatible with all standard host machines with OS Windows 2000, XP or Vista; MacOS; or Linux – was the selected VM technology. To keep the system as general as possible with respect to the various devices and softwares used by countries, the communication between the VM and the host system (on which countries were able to deploy their own third-party application such as case management systems) was restricted to file exchange and minimal script calls.

The internationalization and the management of languages and scripts were of utmost importance in PIAAC. Even if the CAPI screens were designed to trained interviewers solely, it nevertheless would strongly impact the quality of the interview and the collected data. All characters were encoded in UTF-8 (Universal Character Set Transformation Format – 8-bit). The preferred

communication file format was XML, while CSV was also used for import and export, and ASCII text files were sometimes used to exchange small chunks of data during runtime. The translation system was based on exporting and importing an XML-based format called XLIFF (XML Localization Interchange File Format, see <http://developers.sun.com/dev/gadc/technicalpublications/articles/xliff.html>) that is compatible with the usual translator computer-assisted translation tools.

In order to ensure high quality and reliability standards for the collected data, security aspects attached to the CAPI system were thoroughly addressed both in terms of confidentiality integrity and accessibility. Data confidentiality was ensured by full encryption of all data and communication on the interviewer laptop as well as between the interviewer laptop and the consolidation points (at both national and international levels). The internal communications between the components of the VM and between the VM and external third-party applications that could run on the interviewer laptop depended on country requirements. To ensure data integrity and accessibility, a complete crash recovery system was set up. If the system crashed during the interview, it was possible to restart the questionnaire at the level of the last answered question and to provide information to the interviewer about the parts of the questionnaire that were already answered, the last answered question and the active language. The system also blocked most functionalities that did not pertain to the interview to avoid accidental termination of the CAPI program by the interviewer during the interview. However, if accidental termination happened anyway, the recovery system enabled resuming the system and the ongoing interview. Such a system was made possible using a fine-grained and very frequent input data auto-save capability. Auto-save was activated at the level of each input field every time the interviewer selected/entered input in a given question. Data integrity controls were also enabled by providing comprehensive log files of timestamped events and data at both question and flow levels for future external audits.

The CAPI system maintenance was ensured at three different levels: on the host system itself and the VMware software; on the hosted system and the associated software such as the Apache Web server, PHP and MySQL database; and at the application level, together with the configuration and collected data. The last two levels were maintained using versioned VM images and patches provided by the Consortium, while the first level was left under the responsibility of the countries.

Fluency of the interviews was also carefully addressed by providing to the interviewer a highly usable system with standardized and simple interface design, coupled with a comprehensive training of the interviewers. The data input mode was designed to be as easy as possible for interviewers. Indeed, the burden of using external devices such as mice was eliminated by allowing the interviewer to input data, navigate in the question flow, and trigger functionalities exclusively with the keyboard. When answering questions with predefined responses, the interviewer was not forced to sequentially navigate through the list of options. Typing in the numeric code of the answer option on the keyboard automatically checked the corresponding value. Open questions with alphanumeric inputs worked the same way. Predefined standard answers such as refusal to answer or “don’t know” answers were made accessible using shortcut keys. In a similar fashion, triggering functions such as: pausing or leaving an interview, introducing comments and remarks, requesting help from the system or about a question, or navigation functions through the question flow (such as going backward and forward) in the questionnaire were also made by pressing shortcut keys on the keyboard. Along with some layout skins and question presentation conventions, the definition of these keys and shortcut mappings

were made configurable for countries willing to customize them following their own interview standards. The overall design of the user interface was created with input from PIAAC participants in Australia, Canada and the United States.

In many situations, the respondent spoke more than one language with various fluency levels. Within the scope of PIAAC languages decided by countries, the system allowed the interviewer to switch to another questionnaire language that best suited the proficiency level of the respondent, independently of the system interface language used by the interviewer. Similarly, the interviewer had the ability to modify the interface language according to circumstances.

7.2.2 Running the questionnaire

The PIAAC CAPI system enabled the definition and execution of complex questionnaires that included conditional paths depending on previous answers provided by the respondent and a series of adaptive features that supported the interviewer during the whole interview process. Central to the design was the need to keep the interviewer focused on asking the question and collecting the answers from the respondent swiftly. To achieve this, strong support from the system had to be provided to avoid distracting the interviewer with unnecessary system manipulations, controlling the flow of questions, and ensuring response consistency by providing a series of automatic features. The BQ and the system were thus designed to include straightforward question types, navigation facilities, consistency checks, precalculated answers, adaptation of sentences to be read depending on the respondent qualities and previously collected information, and contextual information display and gathering functions.

Figure 7.1: Screenshot of a multiple-choice question illustrating the main questions components

The screenshot shows a software interface for a questionnaire. At the top, a status bar displays 'EN FR | Languages : EN FR | User ID: piaac | Pause | Export | Logout'. Below this, there are two buttons: 'Write a remark (F4)' and 'View calendar (F7)'. To the right of these are 'Back' and 'Forward' buttons. The main question area is titled '[A_Q04cca] Do you speak any other languages on a regular basis at home?'. Below the question, there is a section labeled 'INTERVIEWER:' with instructions: 'Mark all that apply. However, if the 'No' category is selected, respondent cannot choose any other category.' There are four radio button options: '< 01 > No', '< 02 > Yes (English)', '< 03 > Yes (French)', and '< 04 > Yes (other - specify)'. At the bottom left, there is an 'Input Code' field. At the bottom right, there are 'DK' and 'RF' buttons. Navigation arrows '<<' and '>>' are located on the left and right sides of the question area.

The BQ included several classical types of questions in terms of collected values, association with variables and in terms of layouts. Each question was associated to one or more variable. Depending on the nature of the variable, the associated value was a numeric response code, possibly associated to an alphabetic string or piece of text, or a numeric string. Whenever the range of possible value was closed and predetermined, a series of possible answers was provided along with their corresponding response code.

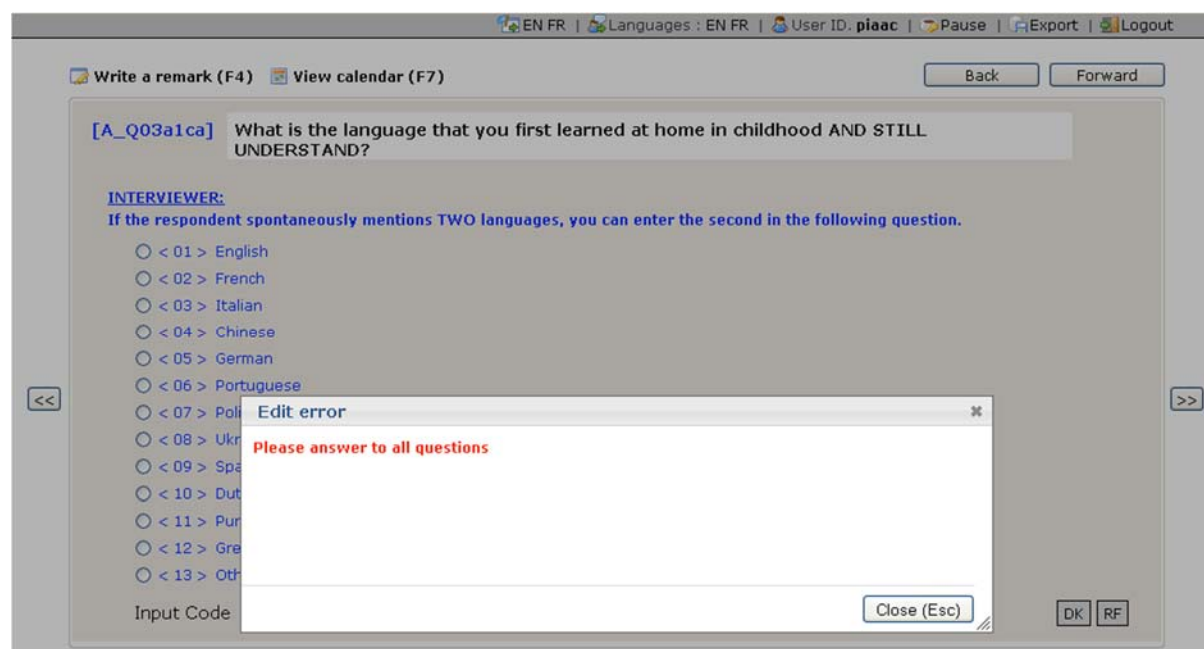
The system accepted both single-choice and multiple-choice questions, as illustrated in Figure 7.1. On the top, the question code was displayed along with the question texts to be read by the interviewer. Underneath the questions displayed instructions to the interviewer in a standardized color code (blue) indicating that the text should not be read. The series of options were provided

with their response codes, among which the last one would be further specified. The globally predefined answers were displayed as the “DK” (don't know) and “RF” (refused) button associated with keyboard shortcuts. Navigation buttons were located on the left (back) and on the right (forward) of the screen. Response codes and navigation actions could also be entered using keyboard shortcuts.

Open questions collected textual or numerical answers. In many cases, free-text entries were used in conjunction with the fixed predefined answers to enable expanding the predefined list with new answers. In addition to the question-level coded answers, the system provided a series of globally defined answers to collect nonresponse for refusal or because the respondent did not know the answer. Responding to some questions was optional, but usually most questions were mandatory.

Figure 7.2 shows how mandatory questions were managed. Whenever more than one variable was associated to a question, various layout were available, among which was the table or array presentation. When meaningful, related questions scan were kept together in question blocks that were displayed together on a single screen. The visual layout was standardized with two basic display types indicating to the interviewer if the text should be read loudly as is to the respondent or if the text was an indication to the interviewer that it should not be read to the respondent.

Figure 7.2: Screenshot of a mandatory question



The question sequencing in the PIAAC BQ followed a complex set of branching rules based on variables, either collected or calculated during the interview, loaded at initialization, or set up as global constants. Branching rules consisted of logical expressions that triggered a jump to a target question when a condition specified as a logical expression on any single variable or a combination of them was evaluated as true. Contrary to some other well-established CAPI systems, the one built on TAO for PIAAC did not evaluate the rule prior to displaying a question as a display condition or a skip rule. In the PIAAC system, the branching rules were always evaluated after the

question was answered to determine what question to display next. In the absence of routing definition, the system assumed a linear sequencing and proceeded to the next question in the questionnaire definition order. Depending on country adaptations, the question flow and routing definition differed substantially from the flow of the international master BQ.

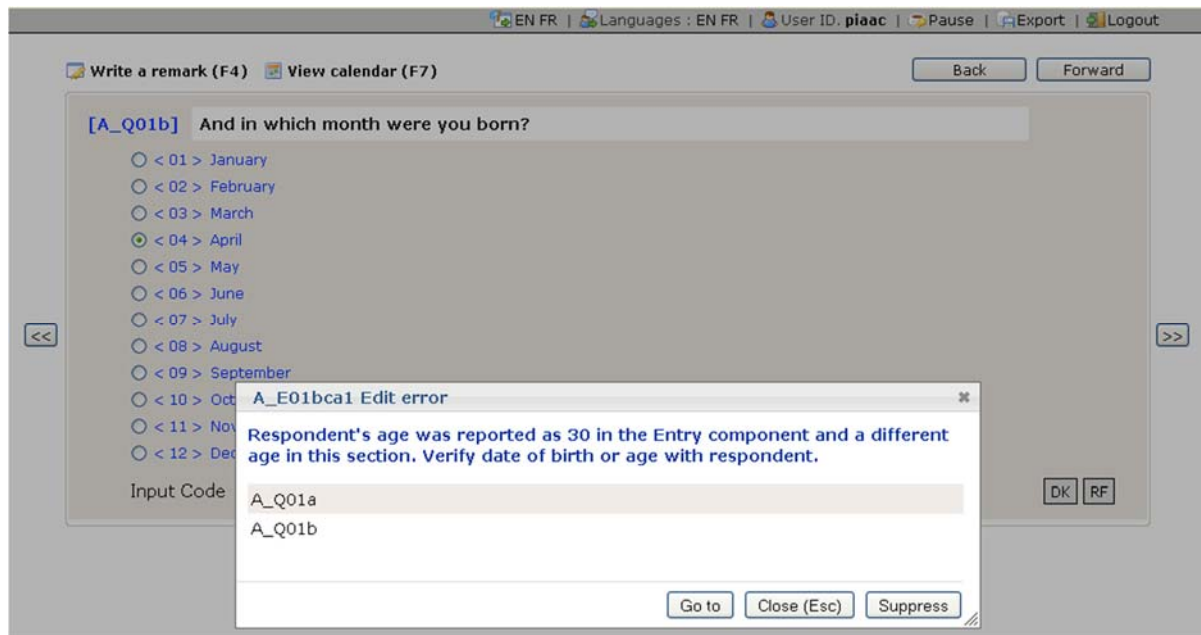
In case the respondent made a mistake or wanted to revise one of his or her previous answers, or when a consistency check forced the revision of a previous question, the system enabled going backward in the interview to modify the response. Such operation was tracked for later quality audit, and, when proceeding forward again, the routing was evaluated again with respect to the new data. Whenever the new question path differed from the previously explored one, all impacted values were invalidated but kept in the database.

Figure 7.3: Screenshot of a numeric entry to which is associated a range constraint and a consistency check

The screenshot shows a web-based interview interface. At the top, there is a navigation bar with links for 'EN FR', 'Languages : EN FR', 'User ID: piaac', 'Pause', 'Export', and 'Logout'. Below this, there are two buttons: 'Write a remark (F4)' and 'View calendar (F7)'. To the right of these buttons are 'Back' and 'Forward' buttons. The main content area displays a question labeled '[A_Q01a]' with the text 'Can you please tell me in which year you were born?'. Below the question is a text input field labeled 'Year:'. To the left of the input field is a double left arrow button '<<'. To the right of the input field is a double right arrow button '>>'. Below the input field is an 'Input Code' label followed by a small text box. To the right of the input code box are two buttons labeled 'DK' and 'RF'.

In addition to the internal consistency of variable inputs that were defined as constraints (such as numeric ranges or string length – for instance, when asking the birth year of the respondent as illustrated in Figure 7.3 – where the numeric value must be in a certain range), the consistency of responses with respect to context variables or previously collected data was maintained throughout the interview to avoid contradictions. These checks could be defined at the level of each question if needed. They were defined with a logical expression based on variable values and state, which triggered, if evaluated as true, the display of a piece of text explaining the nature of the problem and the set of previous questions that contradicted the current answer. Figure 7.4 illustrates the detection of a consistency violation concerning the age of the respondent. The BQ definition enabled specifying if corrections were mandatory or not. In the former case, the interview could not go further without eliminating the contradiction in either the previous or current responses. Similar to branching rules enabling the calculation of question routing, the consistency checks were evaluated after the question was answered before proceeding to the next question.

Figure 7.4 Screenshot of an edit screen appearing after violation of a consistency check



In some cases, the value of variables could be inferred from previously collected data. These situations were defined using inference rules, or auto-filling rules. Such rules were built on a logical expression based on variables that triggered the assignment of another variable if the expression were evaluated as true. The calculated value was obtained using a combination of string and arithmetic operators on other variable or constant values. Because inference rules were independent of the routing, auto-filled variables were sometimes needed irrespective of the previous step in the flow. Therefore, such rules could be evaluated before or after the display of the question to which they were associated. The value of the inferred variables could be used subsequently in any routing, consistency check or adaptive text.

Adaptive texts were specified as dynamic text rules that enabled substituting strings or substrings using variables or conditions on preexisting data. Adaptive texts were used to display the texts for the interviewer in the exact form they were to be read, taking into account the precise context of the interview. They were intended to prevent the interviewer from making on-the-fly adaptations of the discourse and maintaining good fluency of the interview. Typically, the adaptation concerned temporal context, taking into account the situation of the respondent (displaying “When you were at school” if the respondent left school, instead of “At school” if the respondent was still a student), or took care of the gender and, in some cultures, of the polite address depending on the respondent’s age.

Dynamic text provided a specific format in order to specify which part of the text had to be replaced or not, following a set of conditions. For instance, if the gender was coded as female, some parts of the question text had to be adapted accordingly. In the BQ, and in order to maintain the clarity of the text, the definition of the rule was independent from the string substitution. In the PIAAC system, only a variable was included in the text constant as a placeholder for the substitution. The content of the variable was obtained using an inference rule that was typically

evaluated a priori, that is, before displaying the question. While all other rules were only dependent on country adaptation and related to the structure of the questionnaire, dynamic texts modified the content of the questions and were also language-dependent.

Such dependency had several consequences: The localization of the placeholder on the constant part of the text could vary from one language to another (according to the structure of the sentence), and some dynamic adaptations could require a different number of variables in different languages (due to conjugation for example). In principle, for a given piece of text to be translated, the definition of the required variables, their localization in the text, and the specification of the rule condition was part of the translation and should have been made by the translator. However, most translators were unfamiliar with the formal aspects of inference rules and string substitution. An easier solution was put in place where the entire piece of text was changed according to the rule for a predefined set of situations. For each situation (defined in the form of a hidden predefined rule), the translators were asked to proposed complete texts for all variations. The translated files were then post-processed to generate the correct inference rule and substitutions in the BQ definition.

Besides the content of questions, predefined answers and response codes, or rules, the system also enabled authors to define instructions related to the questions. Such instructions were located on the screen according to the reading flow of the interviewer – before the question text, after the question text, and after the response area. The objective was to facilitate the interviewer reading down the screen and getting the instructions in order related to what he or the respondent had to do. For example, an instruction directing use of a show card was given before the question text. This prompted the interviewer to supply the respondent with the show card before proceeding to reading the question. The most common instructions, directing the interviewer on what to do to complete the question, came after the question text. For example, a special coding or probing instruction may have fit in that location. Having read the question, the interviewer received instructions on how to handle input or question follow-up in position with its relevance. Instructions after the responses were special to guide interviewers on what to do next. Rich text was used as a way to control the visual presentation of the instructions for the interviewers.

In addition to on-screen instructions, the system also enabled BQ authors to create supplementary helps and instructions that could be consulted optionally by the interviewers. Such helps usually consisted of more precise definitions of some concepts mentioned in the questions that the interviewer could use to give precisions to the respondent. The question-related help material was made visible in a modal window that popped up when the user requested it.

7.2.3 Running the interview

The CAPI system could be initialized by importing a set of predefined variables, for example, personal data like name, address, and so on of the respondent. Depending on the countries, these data were passed by the basic case management system provided with the PIAAC system, or generated from a third-party more sophisticated case management system installed on the laptop. When the questionnaire was executed, variables corresponding to initialization data were prefilled and usable in rules expressions or visible as prefilled responses to questions. Upon country adaptation, the prefilled responses might then be verified with the respondent and corrected as the case might be. In case of a breakoff, the questionnaire was populated with responses from the previous session when the interview was resumed at some later time.

During the interview, the questions were displayed according to the sequence defined in the BQ. A priori inference rules were first calculated, if any, and the new variable calculated. Then the text substitutions were executed to compose the possible dynamic texts that might appear in any string on the screen, be it question, instruction, help, predefined answers, and so on. The substitution was also performed each time the interviewer toggled between languages or every time a question was revisited during navigation. The later situation could arise if a previous answer was changed that modified the context of adaptation of a subsequent adaptive text of a question the interviewer was coming back to again. The inference rule could also prefill the answer to the question that the interviewer might change.

Figure 7.5: Screenshot of a checkpoint question that is not read to the respondent but contains only interviewer instructions

EN FR | Languages : EN FR | User ID: piaac | Pause | Export | Logout

Write a remark (F4) View calendar (F7) Back Forward

[A_N03a1ca] Did the respondent mention more than 1 language?

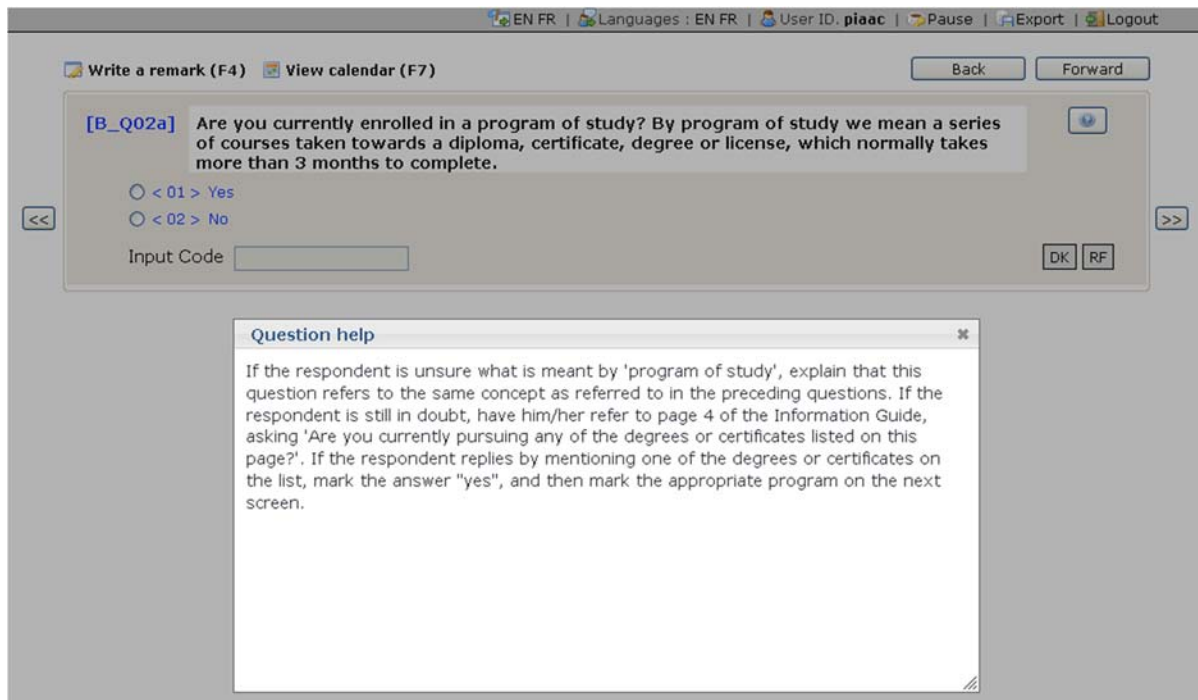
INTERVIEWER:
This question does not need to be asked unless you are not sure.

☐ < 01 > Yes
☒ < 02 > No

Input Code 02

Once displaying the question, the system showed instructions to the interviewer with a specific, regular visual presentation enabling the distinction between different types of instructions and questions read to the respondent. Answers might sometimes be read aloud to the respondent, depending on similar rendering rules as for questions. As illustrated in Figure 7.5, instructions could also be part of special questions that were not read to the respondent but served as checkpoints during the interview. The system also provided help buttons to give access to help information attached to the system, the interview, the current question and the currently used interviewer interface at any moment during the interview. Figure 7.6 shows an example of help material provided to the interviewer.

Figure 7.6: Screenshot of a model screen displaying help material to the interviewer



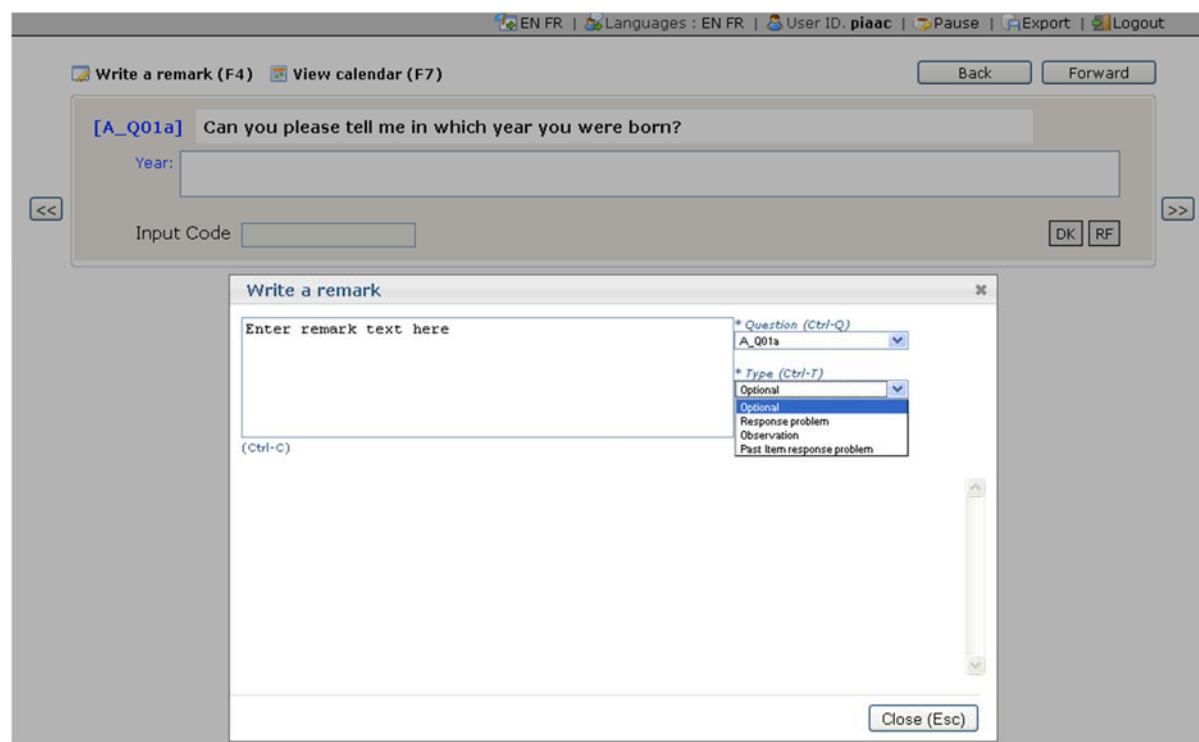
The system checked whether input constraints were met when the interviewer entered inputs with the keyboard. Such verification prevented the interviewer from skipping a mandatory input field or entering values that were not compliant with a mask, a set of acceptable values, a field length limitation, or a numeric range. The check was made on the client side before sending the value to the server. Whenever a constraint was violated when the interviewer validated the response and tried to proceed to the next question, a popup ID displayed explaining the constraint and asking for the answer to be modified accordingly.

After the interviewer introduced the answers of the respondent using the various response fields and codes and pressed the next question key, the system evaluated the consistency checks that might have been specified for that answered question. Whenever the consistency rule was violated, the system produced two types of messages: soft edits and hard edits. Edits are feedback messages presented to the interviewer to report an inconsistent response. They resulted in a pop-up overlaying window with a message to the interviewer identifying the problem and explaining how to fix it. Soft edits identified a range of consistency problems that did not have to be resolved for the interview to continue. In most cases the interviewer was presented with options for resolution that included a means of ignoring or suppressing the edit. In contrast, hard edits identified consistency problems that needed to be resolved for the interview to continue (this included mandatory questions). When edits were resolved, the system checked for the existence of a posteriori inference rules and eventually executed them if necessary.

Once the interviewer instructed the system to reach the next question, if a branching rule was defined, the system proposed the next question according to the branching rule. If not, the system jumped to the next question in the default path, that is, in the path defined by the question definition order.

At any time during the interview, the system allowed interviewers to write down comments about the interface or the question and to modify the comments they wrote. The comment content specified its scope and the diffusion level (private, public, other groups) as shown in Figure 7.7. The system also recorded the event and the timestamp, as well as the author of the comment. In some circumstances, the interviewed needed to be paused by the interviewer and further resumed. Pausing an interview generated a trace in the log for future audit and the interviewer was invited to introduce a comment regarding the interruption.

Figure 7.7: Screenshot of the interviewer comment interface



The interviewer also had the option to terminate the interview at any moment. Irrespective of the fact that the questionnaire was terminated prematurely (by the interviewer or caused by a crash) or normally at the end of the question flow, the system assembled all the log information in a series of output files and triggered the export of them to the host system or to the case management system.

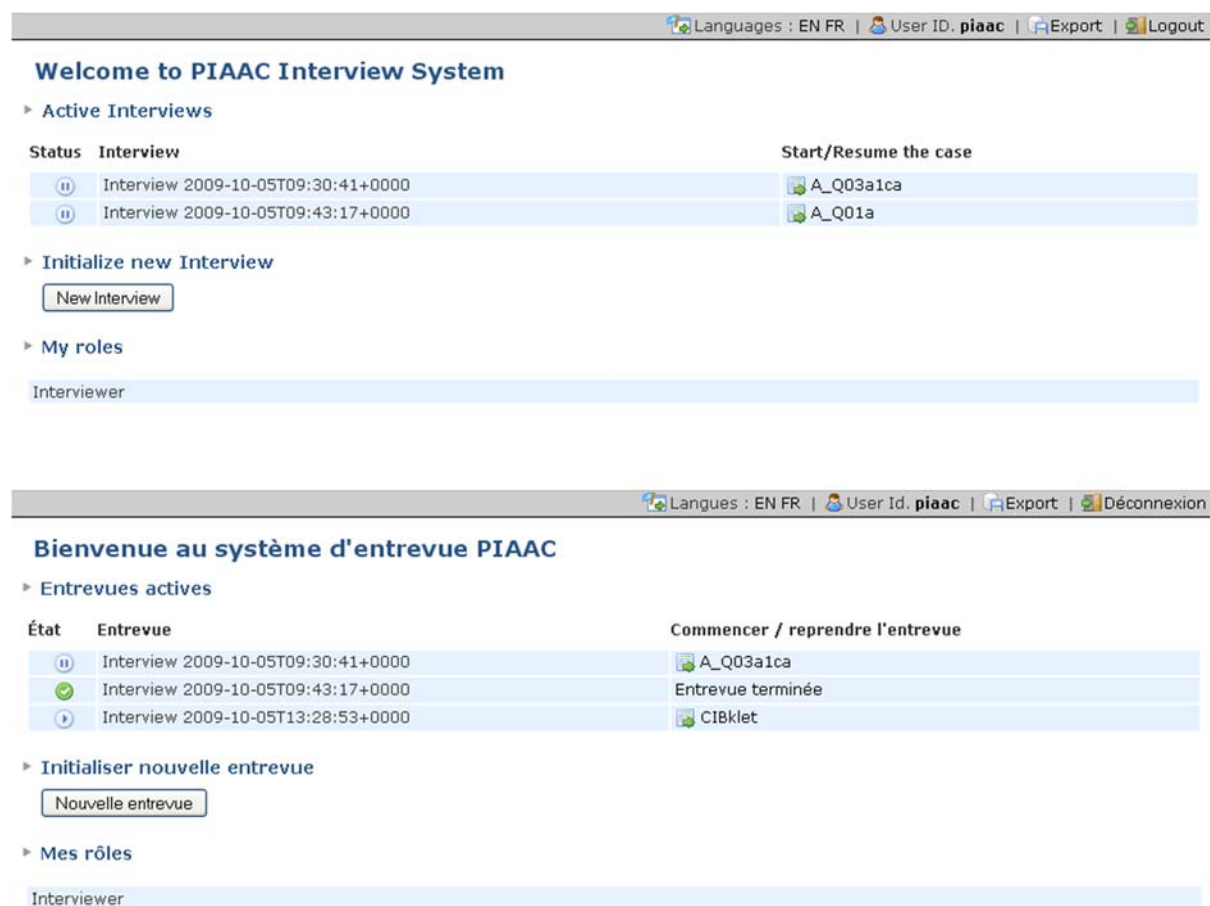
7.2.4 Interview administration

Basic administration features were provided in the PIAAC CAPI system, such as a secured password-based interviewer authentication managed at the country level. The interviewers for a given instance of the system were created by the system administrator. The administrator also assigned and scheduled the interviews for particular interviewers.

The administration features of the system also included a basic CAPI system for countries that did not have or wish to use their own. The case management system managed the interview assignment for the interviewers. It showed them pending cases and upcoming ones and let them start, pause and resume their assigned cases. Figure 7.8 shows the interviewer initialization screen, which was made available in different languages depending on the country language profile. The

case management system also took care of the interview initialization by importing the initialization file as described previously.

Figure 7.8: Screenshots of the interviewer initialization screen in English (top) and French (bottom)



When cases were completed, it collected the termination codes and comments from the interviewer explaining the context of case termination. These codes were included in the export file of the CAPI system.

In addition to driving the data collection for the BQ, the TAO workflow engine on top of which the CAPI system was built was also used to drive the entire process of the PIAAC survey in households. Indeed, a global workflow was created for the whole process that included the specific one defining the BQ and described in this chapter. Such overarching workflow piloted the survey from the very beginning by sequencing the BQ, the ICT-Core components, the routing of the respondent to either CBA or PBA instruments, and the final section for the interviewer. The ICT-Core components and the CBA instruments were delivered as external activities triggered by the workflow.

7.2.5 Support for PBA instruments

PBA instruments were delivered separately in paper booklets handed over by the interviewer to the respondent. However, some instructions were given by the interviewer at some point, for instance, to instruct the respondent to start a section or turn a page, or to provide him or her with a ruler or calculator. Routing rules were also used to manage the assignment of particular booklets to respondent according to the survey design. To each question of the PBA instrument corresponded a question in the workflow engine where the interviewer had to code whether the respondent provided an answer or not, or refused to answer. Provided that interviewers were trained properly, introducing such code in real time while the respondent was solving the tasks, this protocol allowed for recording timing information for the PBA instrument in the same fashion as for the CBA.

7.3 CAPI runtime

The PIAAC CAPI System Runtime component consisted of an instantiation of the TAO workflow engine running within the VM of the interviewer laptop, which executed a series of embedded workflows and activities made accessible through a Web browser and served by local servers and databases. The PIAAC VM consisted of an operating system fine-tuned for PIAAC operations (lightweight and locked, including only the necessary third-party software and system services), a full TAO platform running under this OS, as well as TAO third-party extensions such as the CBA Item Runtime Environment. The TAO platform contained all the necessary information to run:

1. The global PIAAC interview workflow describing the case initialization, the disposition codes, the ICT-Screener, ICT-Core and ICT-Tutorial, the navigation among instruments, the booklet selection controls, and the interviewer instructions for PBA booklets;
2. The BQ; and
3. The cognitive instruments including general and domain-specific orientations.

Among the cognitive instruments, PSTRE was managed entirely by TAO, while reading and literacy instruments were partly delivered by TAO and the CBA Item Runtime Environment.

While TAO piloted the global survey workflow execution of the study, strictly speaking, the part of the system pertaining to the CAPI essentially covered the way TAO was configured to support the questionnaire definition (ontology and related question definitions) and the questionnaire execution (the interview process and the rule system). In this section, we provide an overview of these core elements of the CAPI runtime.

7.3.1 Components

7.3.1.1 Ontology management and question definition: Generis4

As mentioned earlier, TAO is a CBA platform entirely built on knowledge technologies, and particularly constructed around data structured according to the RDF and RDFS standards (Resource Description Framework, 2004) that represent the foundation layers of the Semantic Web. RDF and RDFS, called RDF/S when considered together, are XML-based languages designed to formally describe ontologies. Ontologies are shared conceptualizations of things that exist in a particular domain of interest (Sowa, 2000; Mahalingam & Huhns, 1997). They describe

explicitly the structural part of a domain knowledge in a knowledge-based system using languages with precise primitives (Maedche & Staab, 2001) and associated semantics that is used as a framework for expressing the model (Decker et al., 2000), among which concepts, properties of and relation between concepts, as well as instances of concepts. The formal character ensured that an ontology was machine-processable and exchangeable between software or human agents (Guarino & Giaretta, 1995; Cost et al., 2002). In some pragmatic situations, it simply consists of a formal expression of a metadata framework to describe information units (Kahng & McLeod, 1998).

The kernel of TAO is an ontology management system called Generis4 (standing for GENERIC System for Storing, Structuring, and Sharing knowledge) manipulating RDFS ontologies that structure explicit RDF data and inferred RDF statements according to the RDFS entailment rules, stored in a schema-less relational database. In addition to data-related services, Generis4 also offers a framework to develop Web applications based on Semantic Web technologies, exploiting specific business logics and their associated ontologies in the form of extensions. TAO is such an application based on a series of extensions bearing their own ontologies defining the domain and subdomains of CBA, along with the related application logics and user interfaces. In its basic version, TAO is composed of extensions related to the management and definition of items, tests, test takers, groups of test takers, delivery scheduling, and results export, together with a workflow engine.

As part of TAO, the CAPI system is itself built as an extension exploiting the capabilities of the workflow engine. In the CAPI system, the respondents were considered as test takers bearing a series of characteristics describing the information that the PIAAC survey intended to collect about them and their context – the respondent model and their context form the ontology of the PIAAC survey. Mapping the variables with the characteristic, the collection of the PIAAC data was thus viewed as eliciting knowledge about respondents, that is, a means to instantiate the PIAAC ontology for specific respondents for which precise values were assigned to their characteristics. In TAO, the graphical interfaces to edit the ontologies and their instances were generated automatically from the model. So were the questions displayed in the interviewer CAPI interface, based on a specific question model and templates to display them on screen.

In Generis4, to ease the data and model consultation and editing, users could define specific subparts of the models in the form of hyper classes (they roughly corresponds to views in relational databases). To render the hyper classes for consulting and editing purposes, users could also associate hyper views to them. Hyper classes could be instantiated in the form of hyper instances. A Generis4 service enables one to display hyper classes in edit mode rendered according to its hyper view, and fill in the editable fields to create an hyper instance. In the CAPI system, each question corresponded to an hyper class that was instantiated as a hyper instance for each interview.

In normal use of TAO and Generis4, the different ontologies and their instances were defined using the graphical user interface (GUI) of the system. Performing such knowledge modeling operation requires advanced modeling skills for which PIAAC personnel in countries had low or no competency. In order to facilitate the description of questions and their presentation and execution options, the Consortium provided a specific import/export format of the BQ, together with distinct ad hoc authoring, rendering and management tools. The Consortium took care of importing the files prepared by the countries into TAO.

7.3.1.2 Workflow management and rule system: TAO WF engine

As already explained, the sequence of questions in PIAAC was managed as a workflow where each question represented a data collection activity that fed the corresponding ontology element representing a survey variable. The TAO workflow system consisted of two main parts dedicated to the workflow authoring on one hand, to the workflow execution on the other hand. Depending on country adaptations, the flow of questions might vary significantly between countries. This variability prevented the Consortium from centrally building a single workflow, which resulted in the need for countries to edit their own one. As for the question definitions, defining a workflow was not an easy task and required technical competencies that were not necessarily present in each national team. The Consortium thus provided tools to define the question flow together with the question definitions.

A workflow in TAO was defined as a set of activities sequenced by connectors. Activities are placeholders for services that can be of two distinct types: interactive and noninteractive. The former services correspond to all services provided by TAO through a GUI. Indeed, in addition to the classical GUI, TAO exposes all its atomic management activities, such as item authoring, item metadata editing, and so on as services that can be embedded in workflow activities. Besides TAO functionalities, hyper class services can also be embedded in activities to display forms that feed specific parts of the ontologies. This is the way the questionnaire was executed in the PIAAC CAPI system. Besides TAO services, other interactive services can also be embedded in workflow activities. We used this capability to embed remote item execution from the CBA item runtime in a seamless way for the user.

Noninteractive activities mainly consist of background calculations or calls to distant Web services. In the CAPI system, such activities were used in the process to execute inference rules that precalculate some variables from previously collected data. Noninteractive activities were also used to trigger system functions at relevant moment during the interview, such as the generation of the export file at the end of the interview.

Activities were sequenced by connectors. While in principle, there existed a split (to create parallel flows) and joint (to gather different incoming flows in a single one) connectors, in the CAPI system, none of these were used because the interview only consisted of a single, possibly branched, path. Therefore, only linear connectors bridging one activity to the next and conditional branching connectors were used. The later ones defined possible branches that could be taken according to the evaluation of a logical rule. The conditional branching connectors were intensively used in the PIAAC BQ, as well as to implement the testlet level adaptation of the cognitive instrument. In order to ease the question flow authoring by the countries, the Consortium created a simple rule language that is presented further in this chapter.

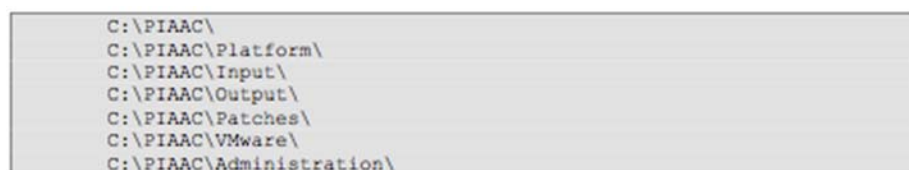
7.3.2 Import, export and interface

The communication between any external software, installed on the host system, and the TAO platform, installed on the VM, could be made either by exchanging data or by triggering services.

All data imported or exported by the TAO platform was formatted in XML files. All XML files were always validated with respect to their corresponding schema described in an XSD file. As a consequence, if an invalid XML file was provided to the system, it was not imported and an error was triggered. Imported and exported XML CAPI variable data files were structured according to

the same XML schema, described in a XSD file. All data that were either imported or exported are located in folder structure on the host system which was replicated into the VM. An automated mechanism ensured the synchronization of both folder systems. The shared folder structure is described in Figure 7.9.

Figure 7.9: Structure of the shared folder between the PIAAC VM and the laptop host system for file exchange



Only three folders were used to administer the VM, import data and export results during field operations:

1. Output. This folder contained all data collected in the PIAAC survey in XML format suitable for exploitation and analysis. While these data could be further reimported into TAO, they were not exhaustive enough to enable restoring a case or a VM in a given state.
2. Input. This folder contained all data that would be imported into the TAO platform when it was launched. These data were either provided by the National Center and copied into the exchange/import folder, or directly written by a case management system and/or a screener installed on the host system as external software.
3. Administration. This folder contained all data enabling case and full system migration or backup, which preserved the current state of the system. These files consisted of database SQL dumps. They were intended to be used for survey operation management only.

The other folders were used for technical maintenance.

A series of services provided in the PIAAC VM could be triggered from the host system. TAO-related services were triggered using an HTTP request with parameters passed via the URL. These services were made accessible from outside the VM using scripts that could be invoked from the host system either manually, or triggered by an external application such as a case management system. VM administration services were triggered using scripts.

7.3.2.1. CAPI initialization

In many cases, the interview started with a series of predefined information relative to the respondent or the case administration (for instance in the Field Test, some random assignments of BQ parts and assessment instruments were predefined and used to configure the workflows for each case), either generated by the case management system or preloaded by the countries in the interviewer laptops. To enable this, an import function was put in place to parameterize the country-adapted workflows (globally or on a case-by-case basis) and prefill variables from registry data or case management systems. Such imported data were placed in the shared input subfolder, which was systematically read and processed by the TAO workflow engine at startup time.

The input subfolder contained at least one global **initialization** file, and optionally a series of **case-related input archives** identified by the respondent's personal identification (PERSID) number and containing one or more variable files.

The **initialization** file was a unique mandatory file present on each laptop. It contained the country-specific TAO behavior parameters and general workflow control variables that were global from all cases to fine tune the TAO system. It thus represented the default invariant value definitions at the level of a country. It was imported each time TAO was invoked, prior to the case initialization data. This data could, however, be overwritten by case-specific data that might be imported afterward using the case initialization file. The initialization file had the same format as the export variable files to guarantee import and export symmetry, meaning that any exported variable could be further reimported as an initialization one or vice versa.

The **case initialization** was made by importing a series of prefilled variables defined in an archive identified by the PERSID of the respondent. As for the global initialization, the file format of the variable files for import and export was identical. The case-level input archive contained the values of all variables to be imported into TAO. These variables could be any existing variable in the BQ, the general PIAAC workflow, the PBA instructions, or the ICT-Core section. Several of these variables were mandatory to start the interview and were referred as the case initialization in the general workflow. The presence of one or more of these files was checked each time TAO was invoked, after having imported the init.xml file. If present, the PERSID variable XML file corresponding to the current interview was loaded with precedence on those taken from the initialization file. The case initialization files could either be written by the case management/screener external system (depending on countries), or directly uploaded from the NC as part of the case assignment, or collected by the interviewer through other means (mails, CD, memory stick, etc.) and copied into the import folder using a script available on the host system. If no such file was associated to a case, TAO then started with the case initialization section of the general workflow.

The launch of the VM could be triggered either on start-up, manually, or from third-party application running on the host system (case management system, screener, etc.). As a Web application, TAO was launched using an URL that could optionally be complemented by a particular PERSID. The presence or absence of the parameter depicted different situations and induced different file import sequences, as well as different user interactions.

Invoking TAO with no predefined case was the simplest method for countries that did not have a third-party client infrastructure to be executed on the interviewer machine. Such a process occurred in two basic situations. First, when the case assignment was made using lists sent to interviewer on a paper format, such as Word documents or Excel worksheets, no case initialization file preexisted in the input shared folder. Then, an assignment sheet provided to the interview with the minimal required information necessary to initialize a new case using the dedicated section of the interview. In the second situation, the case assignment was made by the NC, which sent (by electronic mail; shipment of physical storage devices such as disks, CD, or memory sticks; by download from the NC-secured website; by network-based folder synchronization, etc.) one or more case initialization files in addition to the global initialization one. The case initialization files then contained the minimal required information necessary to start cases. And depending on the global parameterization, the imported case variable or the missing information could be verified or collected through the case initialization section of the interview as the case was.

Skipping the case initialization by setting the global parameterization variable assumed that the required case variable was imported. Those mandatory variables were the PERSID; the respondent's name, age, gender, address, telephone number; the randomly preselected CBA and PBA booklets (depends on whether the system was configured for the Field Test or the main data collection); and in the Field Test, a variable that controlled the 200 rule (see Chapter 14).

TAO could also be invoked by an interviewer by specifying a specific case. This situation particularly arose in countries using their own case management system and possibly their household screener. The initialization variables were managed by these applications that generated the initialization variable file and launched the CAPI system automatically for the specific case. The Consortium also provided simple case management facilities that enabled countries to preload a series of initialization files for several cases. The interviewer started the TAO CAPI system with an entry point that enabled them to select the case to start (or to resume).

7.3.2.2 Administration data exchange during and after the interview

The administration of the machine relative to interviews consisted of the exchange of database dump files between TAO and the host machine, via the case management system if any, or in any other way countries found convenient. The exchange of the dump file went through the administration subfolder of the shared folder. Two dumps could be triggered at any moment during the interview: full system dump and case dump.

The **full system dump** was made on demand by a component from the host system installed by the countries. Its main purpose was to enable the migration of the full system in its current state from one VM to another VM, from one machine to another machine. The target VM or machine would thus be in the same exact state as the source one. The file contained a full dump of the MySQL database where all TAO CAPI data were stored. The dump also contained all the process definitions (the PIAAC survey definitions), the workflow definition, all cases, their paths and variables. The dump file did not contain the cognitive instrument intermediate results and log files, which were managed outside until the booklet execution are achieved. However, the results of completed booklets were exported within the dump.

The **case dump** was also made on demand from the host system and enabled transferring started cases to a different interviewer in its current state from one VM to another VM, from one machine to another machine. The target VM or machine would thus contain the case in the same exact state as the source one. The file contained a full dump of a particular case from the MySQL database where all TAO CAPI data were stored. As for the full dump, the case dump file contained all the necessary information to retrieve a case exactly in the same state as it had been, except for the cognitive instrument intermediate results and log files of an ongoing case, which were managed outside until the booklet execution ended.

All dumps could also be reimported manually into the VM. This operation was not accessible to interviewers but possible only for technicians at the NC level. Dumping the system for administrative purposes was made using a series of scripts that could be invoked from the host system.

The VM and the instance of TAO installed therein provided a series of functionalities to control the CAPI system. These scripts could be called manually or by other programs on the host system. They provided base functionalities to: start the VM with or without specifying a case, at boot time

on the interviewer laptop of triggered by an external application; stop the VM gracefully via normal shutdown of the laptop (crashed VMs could also be stopped to be able to resume normal operation on the next start); start the CAPI system within a running VM with or without specifying a case, stop the CAPI system for a specific case or globally; export results for all or a single case, import all cases that were present in the input shared subfolder or a specific one, dump the cases (all or specific cases) or the whole TAO system; and drop cases for VM cleaning purposes.

7.3.2.3 CAPI result export

Because the TAO workflow drove the entire case, and not only the BQ, the exported files contained all the information pertaining to a case. They were exported in the shared folder between the PIAAC VM and the host system. The export subfolder contained a series of archives, each gathering the output of terminated cases at normal completion of TAO, that is, when the data collection was interrupted temporarily (when the interview was paused) or definitively (when the interview was terminated at the end or before the end) by the interviewer. Each case was exported as a separate compressed file package named according to the respondent PERSID. Depending of the interview completion status, the exported archive might contain zero, three, four or five different files.

The archive contained zero files when TAO exited abnormally and did not export any file. Relaunching TAO triggered the internal recovery mechanism that did not make use of the output files. The partial data could, however, be exported manually using the export services provided in the CAPI system. The archive contained three files: either in case of normal exit from TAO upon premature termination of the interview before or during the cognitive instrument section when delivered electronically; or in case of normal termination of TAO when the respondent was directed to the PBA delivery of the cognitive instrument. The situation when the archive contained four files arose when the cognitive instruments were successfully delivered electronically to the respondent but the post-processing of the raw result file (containing the logs) had failed to export the data properly. Finally, the archive contained five files when TAO terminated normally, with all cognitive instruments delivered electronically to the respondent.

The five exported files per case contained all data collected during the interview and the assessment, together with contextual informations such as the timed log of all events that occurred during the TAO execution and information about the valid and invalidated paths followed during the interview activity flow.

In the archive of a single interview, the unique **Path** file was formatted accruing to an XML schema and contains all information regarding traversed valid and invalid paths along the full PIAAC workflow (including the BQ, and all other steps of the PIAAC survey). The file thus contained the BQ path together with the case initialization, ICT-Core, ICT-Tutorial, ICT-Screener, cognitive instruments and paper-and-pencil instruction related paths. It contained the list of explored question groups (corresponding to the each step of the interview, materialized by a unique screen); for each group of questions, the list of unique questions corresponding to a unique variable; and for each question group, a flag specifying if the explored item group was part of the valid path or not. Interview steps (question groups grouped as atomic activities corresponding to one single screen) covered the whole interview process. If no step of the interview was performed (in case of immediate termination of the interview and normal termination of TAO by the interviewer), the file only contained the topmost XML tags with no question-related information.

The **Variable** file was also unique in the archive of a single case and contained the values of all variables specified in the BQ and the global workflow (according to nationally adapted version specifications) in their final state defined into the TAO platform for a given case. It was formatted according to a specific XML schema and included the data that were imported together, the collected data pertaining to the case initialization, as well as all data collected through the BQ, the ICT-Screener, the ICT-Tutorial, the ICT-Core, and the observations from the PBA instruction sections. The electronic cognitive instrument information was exported in a separated file in the archive. This file constituted the final state of all variables and did not contain intermediate values that might have been changed during the process of the interview. The history of change could be reconstructed from the log file.

In addition to variable names and values, their validity was also provided consistently with the question flow and rules. Hence, all variables that were assigned a value during import or data collection but which finally ended up in a dead branch of the flow, or that were never addressed during the interview, were flagged as invalid. At export time, TAO did not clean the data, enabling the widest range of post-processing possible. In case there were variables for which no value was imported or collected, they were reported using an XML tag with variable name but no value attribute. When the variable file was used for import, if no value attribute was provided, the variable was ignored. On import, all validity attributes were ignored.

The **log** file was formatted according to a specific XML schema and contained the trace of events occurring both at the server side (response of the system to user requests) and the client side (actions of the user on the interface) pertaining to the case initialization, the BQ, the ICT-Screener, the ICT-Tutorial, and the ICT-Core related items, as well as the PBA instructions. All entries in the log were timestamped and enabled to reconstruct the entire sequence of user action and system responses of the CAPI system. An equivalent logging of events was provided within the cognitive instrument result file for further analysis at psychometric level. Except for the cognitive instruments (which were operated by the respondent), all actions of the interviewer were recorded in the log. This file was also used to generate audit trails for verification of interview quality.

Cognitive instrument results were exported in a series of separate files corresponding to clusters of CBA instruments for a given case. The result consisted of the scores (when scoring was made automatically) for all tasks, units, and tests, the PERSID of the respondent, the responses, and other contextual information. It also contained the log of all recorded user actions on the interface. The result file was exported at the end of each booklet execution, as part of the assessment workflow service, before the PIAAC general workflow automatically resumed and proceeded to the following survey steps. Crash recovery data were not stored in this file but in a dedicated structure enabling recovery independently of result export.

The raw cognitive instruments result file was seconded by another post-processed file containing the **CBA Variables** only. The supplementary result file contained the same information as the previous one, with the exception of logs, which were post-processed to provide meaningful additional scoring-related variables, instead of lists of atomic events. It was exported at the end of each booklet execution by the assessment workflow service, before the PIAAC general workflow automatically resumed and proceeded to the following survey steps. It was produced by a post-processing routine that analyzed patterns of events in the raw result CBA file and generated a series of variables that were used for further analysis and scoring (in the case of PSTRE).

The generation of export files was triggered at each normal completion of TAO, irrespective of the case status (completed or not completed). When an interview was started, it could be paused or terminated with a complete case or with a partial case (definitive termination before reaching the end of the interview). When the interview was paused, the result files were exported with the disposition codes relative to the three main survey sections: BQ, ICT-Core and cognitive instruments. In the presence of a case management system, the exported codes could be used in monitoring processes set up by the countries. Whenever the case was prematurely terminated by an interviewer operation, the workflow proceeded to the last activity of the flow relative to the section, ensuring the interviewer was presented the correct disposition code entry screen before ending the interview. At the end of the interview, whether upon premature termination or normal termination with a complete case, the result files were also exported. Resuming a case with the sole exported results was not sufficient. Such operation required other information that was stored in the database.

References

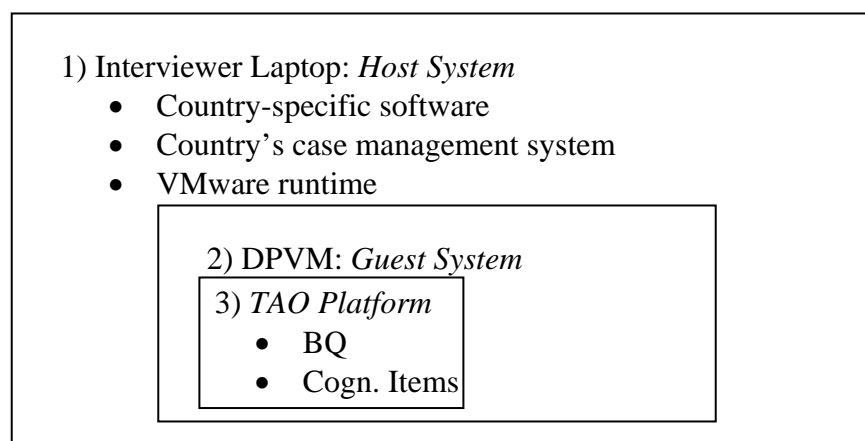
- Cost, R. S., Finin, T., Joshi, A., Peng, Y., Nicholas, C. Soboroff, I., Tolia, S. (2002). ITalks: A case study in the semantic web and DAML+OIL. *IEEE Intelligent Systems*, 17(1), 40-47.
- Decker, S., Melnik, S., Van Harmelen, F., Fensel, D., Klein, M., Broekstra, J., Horrocks, I. (2000). The semantic web: The roles of XML and RDF. *IEEE Internet Computing*, 15(5), 2-13.
- Guarino, N., & Giaretta, P. (1995). Ontologies and knowledge bases: towards a terminological clarification. In N. Mars (Ed.), *Towards very large knowledge bases: Knowledge building and knowledge sharing* (pp. 25-32). Amsterdam, Netherlands: IOS Press.
- Kahng, J., & McLeod, D. (1998). Dynamic classificational ontologies: mediation of information sharing in cooperative federated database systems. In M. P. Papazoglou & G. Sohlageter (Eds.) *Cooperative systems: Trends and direction* (pp. 179-203). San Diego, CA: Academic Press.
- Maedche, A., & Staab, S. (2001). Ontology learning for the semantic web. *IEEE Intelligent Systems*, 16(2), 72-79.
- Mahalingam, K., & Huhns, M. N. (1997, June). An ontology tool for query formulation in an agent-based context. *Proceedings of the Second IFCIS International Conference on Cooperative Information Systems*. Kiawah Island, SC, 170-178.
- Resource Description Framework (RDF) (2004). Retrieved from <http://www.w3.org/RDF/>
- Sowa, J. F. (2000). *Knowledge representation: Logical, philosophical, and computational foundations*. Pacific Grove, CA: Brooks Cole Publishing Co.

Chapter 8: Development of the Integrated Computer Platform

Britta Upsing, Frank Goldhammer, Maya Schnitzler, Robert Baumann, Roland Johannes, Ingo Barkow and Heiko Rölke, DIPF; Thibaud Latour, Patrick Plichart, Raynald Jadoul and Christopher Henry, CRP; and Mike Wagner and Isabelle Jars, ETS

This chapter describes the electronic test delivery system of the PIAAC study. The complete delivery system consists of three parts that were installed on the interviewer's laptop. First, the country's case management system selected and organized the participants in the study and interacted with the embedded virtual machine (VM). As this was country-specific software, we cannot specify the content of these programs, but we can state that a common interface was defined, making access of the PIAAC VM possible. The second part was the VM itself, which served as an encapsulated environment to installed software from other household surveys and to the different hardware configurations existing in different countries. Third, there was the BQ and cognitive items running on the TAO platform within the VM. All these parts interacted; the workflow of interaction is described here in Figure 8.1.

Figure 8.1: Design of the electronic test delivery system



The interviewer laptop ran its normal operating system, the *host system*, which was supposed to be a Windows system (see the technical standards documents). The host system ran optional country-specific software such as a country's case management system and the runtime for the virtual machine.

The virtual machine (DPVM: delivery platform virtual machine) ran within the host system. It is called a *guest system*. It ran TAO, including the BQ and the cognitive items.

The delivery of the BQ and cognitive instruments was done by means of TAO, which ran within the VM. In doing so, there were a minimum of dependencies, influences and interferences between the PIAAC delivery system and the interviewer laptop.

8.1 Development of the virtual machines

The VMs containing the PIAAC system and items are described here in more detail.

8.1.1 VM basics and hardware

All VMs delivered in PIAAC were based on the same prototype VM system. The technical details are as follows:

- VMWare Workstation 6.5-7.x virtual machine
- Virtual hardware
- 1 GB RAM
- Single core processor
- 40 GB HDD
 - Dynamic allocation
 - 2-3 GB actually used
- Display 1024 x 768 pixel

8.1.2 Content of the PIAAC VM

The operating system running inside the virtual machine was a Debian Linux system. Debian ensures a high level of dependability combined with the assurance that no licensing is necessary (open source strategy). Unnecessary software of the standard Debian distribution was removed to save space. This is an overview of the software components used in the PIAAC VMs.

Operating system:

- Debian Lenny
- Kernel 2.6.26-19 – Aug 2009
- X Windows
- IceWM Window Manager

LAMP stack

- Apache2 2.2.9-10

- PHP5.2.11-0.dotdeb.1
- Suhosin-Patch 0.9.7
- Zend Engine v2.2.0
- MySQL v14.12

8.1.3 Development process and automation

There are different parts of a VM. The base is the “Mother VM.” It contains all common data, like the operating system and all software components, which were independent of what country used it. This VM needed to be supplied with all country-dependent information and data.

The Item Management Portal (IMP) held this information. Each country had an authorized person adopt items and translations.

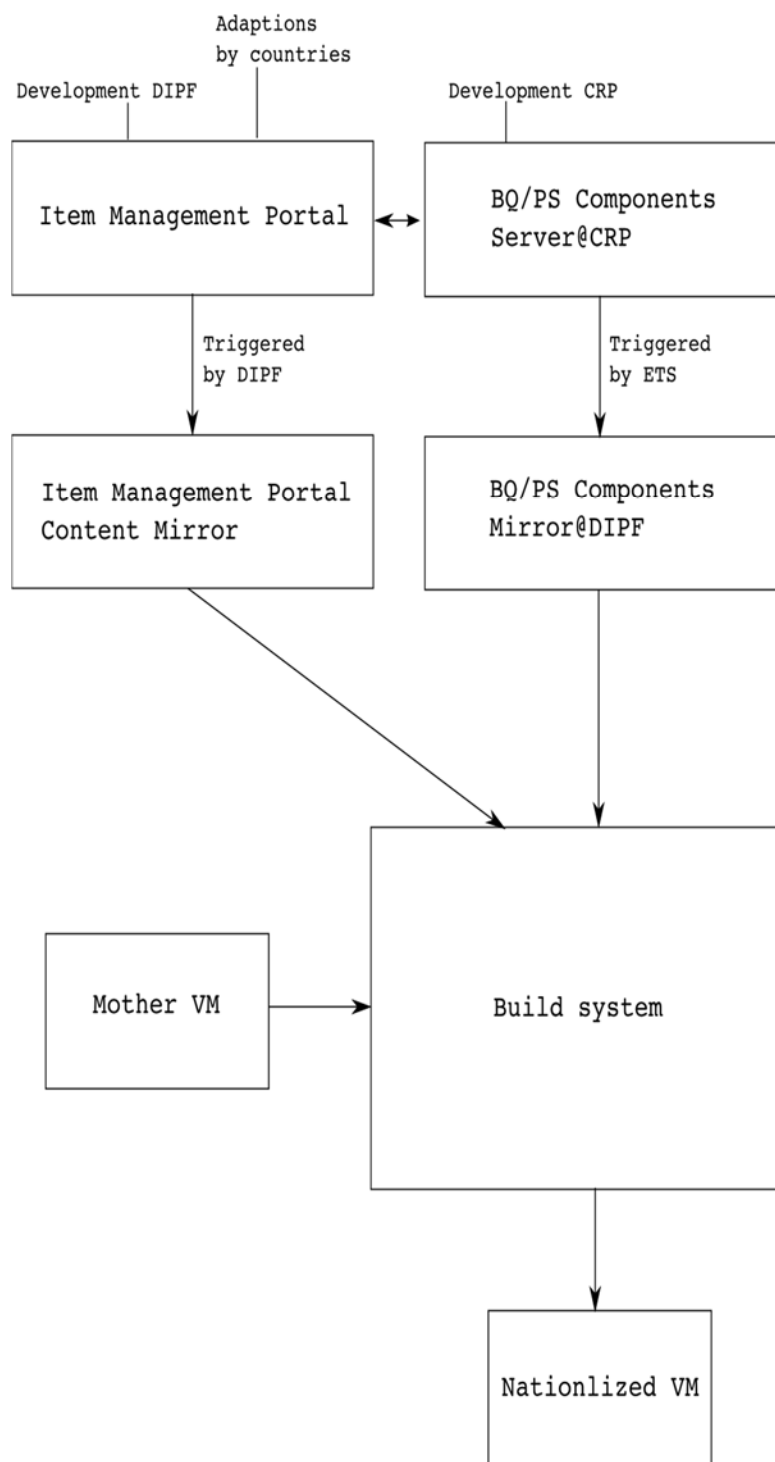
Several servers from the Consortium partners provided all necessary data to the IMP.

At dedicated points of time or after important updates, two main servers were mirrored for the build process. This helped get dedicated versions with timestamps.

The build system fetched all data from the mirror sites and combined this with the Mother VM.

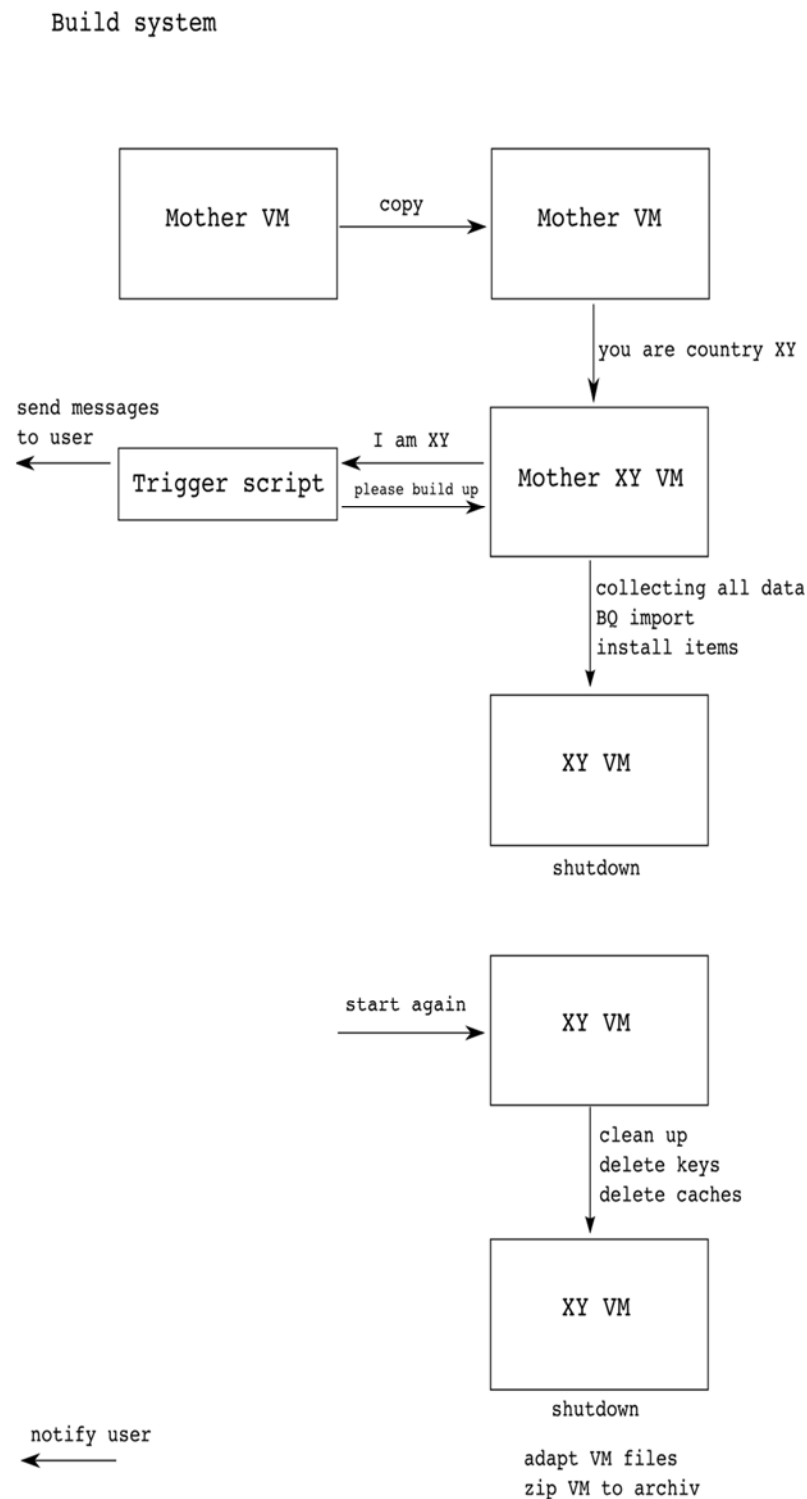
This process is denoted in Figure 8.2.

Figure 8.2: Development process and automation



The build system (Figure 8.3) was a collection of scripts, based on the Bourne again shell (bash), on a build host.

Figure 8.3: Build system



The process was triggered by a bash script for mass production. A loop would call for one task, which will be described as follows. A template of a Mother VM was copied to a new folder. The build system started this copy. The VMware VMPlayer software needed an environment to place a window. A virtual frame buffer was installed and all graphic output was dropped there. It is important to know there was no user interaction possible or necessary. Installing a new Mother VM needed a start by hand to ensure there were no error windows blocking the process. The VM received a country ID as its name in the configuration file. The VM reading this environment variable and contact a script outside via secure shell (ssh).

This trigger script controlled the process from outside. The advantage of this concept was that the messages of the inner scripts were passing the trigger script and could be sent to the originating user. The trigger script sent a start mail to the originating user and called a build script inside the VM. The IP of the VM was transmitted when the trigger script was invoked. This inside script was responsible for several tasks. It fetched all data from the mirrors, did a BQ import and configured all locale settings (keyboard, fonts, etc.)

After this, the VM performed a shutdown, and build task started it up again. This second start invoked the cleaning scripts inside. This was done after a reboot, providing the chance to suppress the second boot up for debugging purposes.

Before the VM did a second shutdown, all installation scripts were disabled and a reboot brought up a nationalized VM.

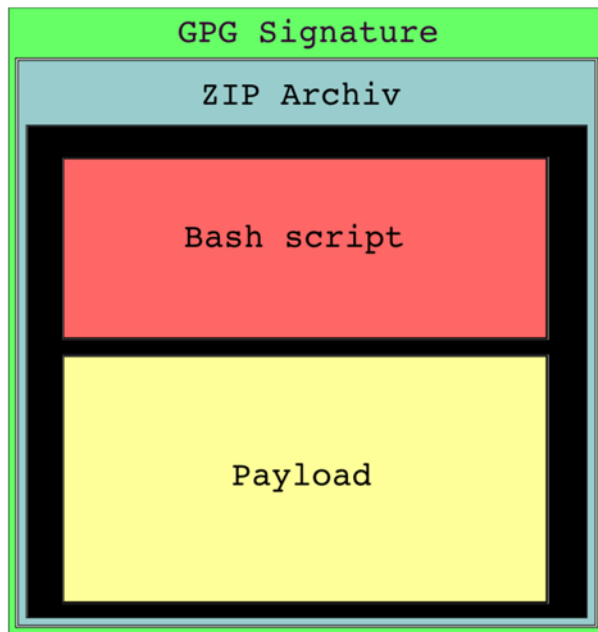
After shutdown, the build script adapted the VM files. Inside the VMX configuration file, some parameters were changed and the caches and the snapshot configuration file were deleted. Four files remained in the VM subfolder. The last step was to zip the folder and send a notification to the originating user.

8.1.4 VM patching process

The intention behind the patch mechanism was to have a robust way to modify the system in any manner (see Figure 8.4). Maybe it would even be necessary to modify the patch mechanism itself. To install a patch, you needed to place a file at a dedicated place inside the VM. This could be done with a graphical user interface provided by the Windows-scripts managing software or via a command-line interface with secure copy protocol (Linux, use WinSCP or the country-specific Case Management System (CMS) Tools under Windows).

The “init” scripts, which were called in the boot process of Debian, had to look for patch files. For these purposes, the init script “/etc/rc.local” was modified. If it found files in “/var/www/piaac/Exchange/patch,” it then checked each file with a GPG signature, unpacked all files inside each zip file and executed a bash script, found inside each patch package.

Figure 8.4: VM patching process



A patch file contained an executable bash script and a payload. The payload consists of additional files that represent the fix itself. These were copied to the destination folder by the bash script. This mechanism allowed many things to occur inside the VM such as kernel changes, exchanges of tests, updating software and many more. This involved considerable risk if you provided a patch file with dangerous contents. There was the possibility of completely destroying the VM. To ensure applying a correct patch file, it carried a correct GPG signature. The keys for signing were different between the Field Test and Main Study, so you could not install a Field Test patch in a Main Study VM. Also it was impossible to install a patch from anyone but the PIAAC Consortium.

After executing the patch bash script, the patch file was moved to the installed subfolder. The next reboot would not be able to see old files and try to execute them.

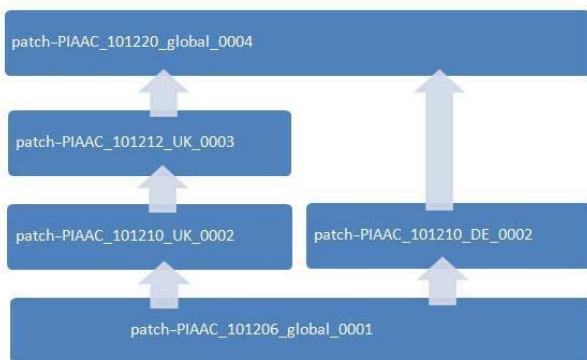
For building new patches, a patch-build VM is useful. This could be an “old” Mother VM with the complete subfolder structure and all keys for signing. It was possible to build patches under Windows, but the end-of-line characters in text files were different between Linux and Windows. The patch mechanism would be confused by Windows text files. If you used “Notepad+” for editing, you could configure this feature to overcome the problems. Other issues were ownership and rights of the files inside a patch. Windows users needed to ensure the files would be owned by root and that the bash script had execution rights. It was recommended to use Linux or a patch-build VM for highest efficiency.

The content of a patch file was simple. It contained at least two files. The bash script needed to be called “execute.bash” and have a file called “description.txt”. The bash script is explained

above; the description.txt file contained a short description of the function – the involved files and fields about target countries, provider, builder and date. The content of description.txt was shown after reboot of a fresh patched VM in a window. This was managed in the IceWM startup file.

The patches were created in a numbered order. Because one patch could be dependent on another, it was necessary to install patches in a correct order.

Figure 8.5: Naming of patches



8.1.5 Changes from Field Test to Main Study

The build chain for the VM was only bug-fixed and some minor changes were done. More modifications were applied to the patch mechanism. To avoid the potential problem caused by trying to ensure patches were installed in the order they were released (as during the Field Test phase), for the Main Study there was no need to take the order into consideration.

For the Main Study, every new patch included the previously released ones (cumulative patch, see Figure 8.6). The VM would know about the so-called patch level, which meant that every VM held a list that contained the patches already installed. Patches already installed were skipped during the installation process.

National patches were included in the cumulative patch. A minor “problem” with national patches was that they only increased the patch level and did nothing else for countries that were not involved.

The naming of the patches for the Field Test was handled individually. For the Main Study the naming was as follows (see Figure 8.5):

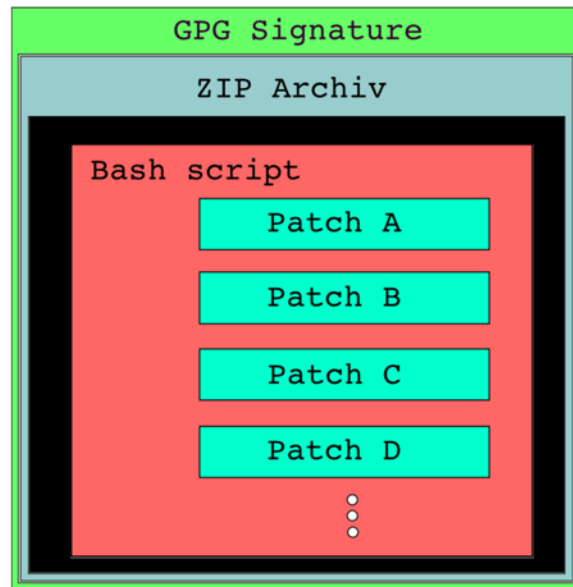
patch-PIAAC_<date>_<responsibility>_<number>.zip.gpg

where **<date>** : the release date, format : yymmdd

<responsibility> : was “*global*” for patches to be installed by all countries or will be the country code for national patches (e.g., “*UK*”)

<number> : the release number starting at “1”, format nnnn. The number was increased incrementally each time a patch was released.

Figure 8.6: New design of patch mechanism in Main Study



An exception was that national patches released at the same time for the same purpose but for different countries had equal numbers. For example:

- Both national patches released on 10 December 2010 included the global patch released on 6 December 2010.
- The national patch released on 12 December 2010 included the national patch for England/Northern Ireland (UK) released 12 December 2010 and the global patch released 6 December 2010.
- The global patch released on 20 December 2010 included all national and global patches below.

It was possible that during a phase of PIAAC the cumulative patch file would get too big because there were too many files to replace (e.g., computer-based assessment projects for all countries). It wasn't expected to happen and did not happen, but still there was a strategy to handle this case:

If a patch needed to replace a large number of files or huge files, these files were provided in an external signed file. The user needed to download the patch file and the supplementary file. So this file was required only once. Subsequent patches did not include the supplementary file's contents but would require it if the previous patch wasn't already installed.

The patch itself was extended into the Main Study. In simple words, it contained one patch as with the Field Test, which contained all other patches. The superior patch executed a bash script as usual. But this script coordinated the installation of the contained patches. It checked the patch level and decided which patches should be installed. For each contained patch, there would be the normal patch installation mechanism operating. Each contained patch had the same structure as a Field Test patch (execute.bash, description.txt and payload). A cumulative patch was a meta-

patch, which executed all other patches inside in a correct manner. In other words, a Main Study patch was a Field Test patch, and the payload was a bunch of patches.

8.2 Development of the interfacing software for the integration in a national CMS (DIPF/ETS)

The VMs running the PIAAC software were well insulated from the surrounding host system. In fact, this was the main requirement leading to choosing VMs as the building block of the PIAAC delivery. Nevertheless, restricted communication between host and guest (VM) had to be possible, for example, to (re-)start or stop the VM and exchange data such as results or patches. Normally, a so-called case management system (CMS) ran on the host system supporting the PIAAC interviewer by managing the sample and the interview status. The CMS could use the interfacing software as a kind of remote control for the assessment software in the VM.

8.2.1 Interface software requirements

Requirements resulting from the environment were as follows:

- Because Windows (XP, Vista, 7) was used, the interfacing software was to be designed to run under these operating systems.
- The participating countries did not use a common CMS, if any. This required that the software be accessible from any CMS software and not require a special runtime environment.
- The PIAAC VM could be run by three different VMware products, so the interfacing software needed to support these products :
 - VMware Workstation (mainly used for testing and setup purposes)
 - VMware Server (used in the Field Test)
 - VMware Player (used in the Main Study)

The functional requirements for the interfacing software were:

- VM remote control
 - Starting the VM
 - Terminating the VM
- Assessment control and data access
 - Start a case
 - Resume a case
 - Getting a case state

- Retrieving the result data
- Handle maintenance and administrative requests
 - Install patches
 - Archiving and recovery

8.2.2 Implementation of the interfacing software

The interfacing software was developed using “AutoIt,” which is a freely available programming environment for Windows. All releases of the software also include the complete source code. This enabled the countries to make changes to the source code if necessary without the need to buy expensive programming tools.

The functionality mentioned above was developed as small programs – the so-called PIAAC scripts. These scripts could be run from the command line or could be called by a program (e.g., a CMS).

The scripts interacted with the PIAAC VM via the VMware VIX interface software. The VIX software provided services such as controlling VMs, file handling and calling programs and scripts inside the VM.

8.2.3 Setup

To install the PIAAC scripts, the scripts were downloaded and copied to a particular location on an interviewer computer.

To handle different user environments concerning the location of the files (scripts and virtual machine) and settings like VMware access information, there was a configuration file which could be modified by a configuration script.

The required folder structure was:

- C:\piaac
holds the PIAAC scripts (this folder can be changed)
- C:\piaac\input
holds the prepared input files containing case data
- C:\piaac\output
holds the result files of finished or paused cases
- C:\piaac\administration
holds supplementary files like database dumps
- C:\piaac\patches
holds patches to be applied to the virtual machine

8.2.4 VM remote control

To control the VM remotely, two scripts were developed. Script StartVM, which was developed first for the Field Test, started the VM. This functionality moved to the script HandleCAPI, which was developed later for the Field Test. This script started the VM only if it wasn't running yet and couldn't start an interview.

The script StopVM forced the Debian operating system to shut down and terminate the VMware software.

8.2.5 Assessment control

For the Field Test script, StartCAPI handled the start of a case and script ResumeCAPI handled resuming a paused case. Both functionalities in the Field Test later moved to script HandleCAPI, which was able to determine the state of a certain case and was dependent on the state to start or to resume a given case.

Starting a case required an input file on the input folder. This file was copied inside the VM, and after that, a service script inside the VM was called that started the browser and displayed the interview at the state specified by the input file (new interview) or at the state it was paused (already started).

A CMS then could look for the state of the running interview by frequently calling the script GetCaseState. Every call of GetCaseState produced a file containing some data about the state and the progress of the current interview. This file was located on the administration folder. If the state changed from "running" to "paused" or "finished," a CMS could react accordingly:

In case an interview finished, the file containing the collected interview data could be copied to the output folder by calling script ExportResult.

A CMS then could start or resume a new interview or terminate the VM by calling script StopVM.

8.2.6 Data access

The script ExportResult was used to retrieve the collected data of a certain interview or all interviews. The file or the files was copied from the VM to the output folder. Inside the VM, the file was moved to an archive folder. This was to avoid having a result file copied more than once.

Script ControlCAPI frequently called the script ExportResult, so every time a result file was written, it was copied automatically to the output folder.

8.2.7 Maintenance and administration

Dropping cases

Deleted the interview specified by the PERSID from the TAO database. This operation was irreversible and needed to be used with care. There was no recovery for this action.

Recover case data

Recovered the result files of a certain case or of all cases from the database or from archive folders and copied the files to the Windows environment. The files were not copied by default (see below). Recovering from the archive folders was preferred to get complete result files.

Dump

To get a dump of the SQL database containing the case data of a certain case or of all cases, there were two scripts named DumpCase and DumpAllCases. The SQL dump files were copied to the administration folder.

To import previously generated SQL dump files to the SQL database inside the VM, there were two scripts named ImportCase and RestoreAllCases. The DisableKeyF9 disabled the debug feature “Watch window.” After a VM was tested successfully, this script needed to be executed one time on a clean VM to turn off the debug feature. Note that the feature could not be enabled again.

Patches

If a bug in a VM was discovered up to now, the bug was fixed and a new VM was provided for download. Downloading a new VM meant it would take time until the new VM was available on the target system. Also, because VMs are so large, it could be problematic to deploy new VMs once interviewers had started work in the field. An easier and faster way to fix a bug in a VM which was already installed was to provide just the changes needed in the form of a patch.

A patch is a small file that is provided for download and contains only the changes to fix certain bugs. Because only the changed files, not the entire VM, were included in the patch, transmittal and installation was fast. This saved time. Installing a patch was secure because the files were signed. A VM would not accept a patch file that wasn’t signed or was signed with an invalid key.

There were two methods for installing patches, either via the command line or via a graphical user interface (GUI). For manual patch installation, the GUI version was easiest. For automated installation, for example, via a CMS, the command line version was best.

8.2.8 Interface overview

Scripts were to be called from the command line in a Windows environment. Some of the scripts required one or more parameters, for example, the PERSID, to specify a certain case. One thing to avoid was starting a script by double clicking on the name, for example, in the Windows Explorer. In that case it was not possible to enter parameter values.

Every script displayed a small piece of information about its purpose and usage if the parameter value “**help**” was entered, such as “**StartCAPI help.**” To manipulate the behavior of a script, switches could be added at the end of the command line (last parameter value). In case of more than one switch, the first character was ‘-’ followed by the switches’ names (usually one character), such as “**-dop**”.

All scripts supported the switch “**-d**”, which produced an editor window for debug output. This debug output could be saved to a file for bug report purposes, such as via the bug tracking mechanisms established.

Configuration tool

- PIAACscriptConfig

Basic scripts

- StartCAPI (optional: “new” “login” or PERSID)
- ResumeCAPI (PERSID)
- ExportResult (optional: PERSID)
- ControlCAPI
- StopVM
- GetCaseState (optional: PERSID)
- HandleCAPI (optional: “new” “login” or PERSID)

Administrative scripts

- DropCase (PERSID)
- DumpCase (PERSID)
- ImportCase (PERSID)
- DumpAllCases
- RestoreAllCases
- RecoverCase (optional: PERSID)
- DisableKeyF9
- PatchVM
- PatchVM_GUI

8.2.9 Changes and enhancements from Field Test to Main Study

During the Field Test it was important to apply the patches in the right order. For the Main Study, the method was changed to make the patch process safer.

Patches were now cumulative, meaning each new release of a patch also contained the patches released before. Each patch now owned a so-called patch level. The patch level was a number that was increased incrementally for every new release of a patch. The patch level was part of the file name (see the last four characters).

If a patch was applied to a VM, it compared the patch levels of that patch and the one within the VM. The patch would be applied only if the patch level was higher than that of the VM. Otherwise it was ignored. It was recommended to apply the patch having the highest patch level if there was more than one patch applicable.

The script PatchVM_GUI released for the Main Study offered only patches having a higher patch level than the current VM.

Chapter 9: The TAO Platform

*Raynald Jadoul, Patrick Plichart, Jérôme Bogaerts, Christophe Henry
and Thibaud Latour, CRP Henri Tudor*

TAO is a platform developed as open-source software. It was initially designed for national education monitoring in Luxembourg, Germany and Hungary. It is also used for many other pilot studies worldwide. The major strengths of TAO (in French “*Testing Assisté par Ordinateur*”) reside in its flexibility and a design oriented toward a multilingual, highly distributed and cooperative operationalization of survey processes led in an international context.

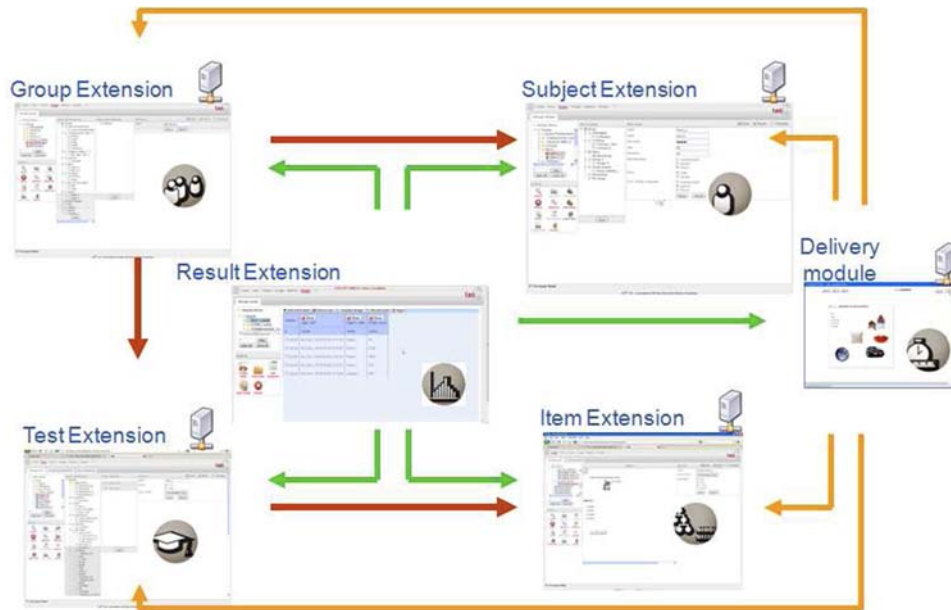
Since 2006, the OECD has relied on TAO for a progressive computerization of its large-scale studies (e.g., the Programme for International Student Assessment, or PISA, 2009; PISA 2012, PISA 2015, as well as PIAAC).

As a technical solution, TAO provides a general and open Web-based architecture for the design, development and delivery of computer-assisted tests. It is released under the GPLv2 license and available on the TAO website (<http://www.taotesting.com>). Although TAO provides much of the functionality required for the operationalization of large-scale assessment processes (authoring tools, workflows for the management of the activities related to the creation of test takers, deliveries, etc.), it was essential to enrich the platform for PIAAC with various features described below.

9.1 TAO architecture overview

In terms of architecture, TAO is built on top of a knowledge base (i.e., a database capable of handling highly flexible data models) called Generis. The TAO platform is 100% Web-based. Its architecture entails independent components (called “extensions”) covering the operations involved in a typical CBA lifecycle (see Figure 9.1).

Figure 9.1: TAO's extensions-based architecture for CBA lifecycle



9.1.1 Item (or question) management

This part manages the creation and design of items. An item can be an exercise, that is, a competency assessment that takes the form of a question for which there is at least one right answer and a scoring algorithm related to the competency to be evaluated. Exercises are also used for formative purposes. An item can be an informational question, meaning a question that does not seek to evaluate a competency but to elicit background information that provides the context (social, economic, etc.) of the test taker. Different design templates are proposed depending on the type of item to create.

9.1.2 Test (or questionnaire) management

This part manages the creation of tests. Tests combine a selection of items into a defined set. Test parameters include item order, scoring, layout, and so on. In the terminology of TAO, a test that integrates solely competency assessment exercises is called a competency assessment (or a computer-based assessment; CBA), and a test that is composed uniquely of informational questions is called a background questionnaire (BQ).

9.1.3 Test taker (or interviewee, or respondent) management

In this part, one can register test takers for the platform, define their registration data (e.g., login, password, mother tongue, location, etc.), and associate them with the relevant group(s) to which they belong.

9.1.4 Group of test takers (or interviewees) management

This part manages the creation of groups for organizational purposes. For example, grouping test takers according to global features and classifications (like the citizenship to a country) is managed in this component.

9.1.5 Delivery (test taking, examination, or interview) management

This part manages test deliveries. The creation of deliveries is the process of assigning selected tests and selected groups of test takers to delivery campaigns. During the creation of a delivery, it is possible to exclude test taker(s) on an individual basis, notwithstanding their group membership. A delivery campaign features many parameters: sequence of the tests, maximum number of executions, delivery period, and so on.

9.1.6 Results management

This part manages the results of the “executions of the deliveries” (also called instances); these data include the information of all delivered tests, their related test taker-, group- and item-specific data, as well as the individual data collected during the “test execution” (also called runtime). Individual data collected during runtime include behavioral information (e.g., reaction times, latencies, hesitation) and contextual information (e.g., hardware settings). In this extension, one can create tables to visualize the results and export them for further data analysis.

9.1.7 Process management

The focus of this extension is the creation and operation of the processes (operated through workflows) required to drive the various types of activities needed in the development of large-scale surveys, hence to support all specialists involved in carrying out those activities.

9.2 TAO architecture for PIAAC

The context of PIAAC required building new extensions and support tools on top of the TAO platform.

First, TAO integrated a new type of item to support the CAPI survey style. CAPI surveys do not deliver the question items directly to the respondent; instead, professional interviewers read the questions to interviewees; questions are backed by instructions and complementary information that guide the interviewers. From a technical point of view, this new form of question item was a major challenge; it had to transpose all the requirements and expected capabilities of standalone (i.e., not Web-based), highly responsive, fully-fledged commercial CAPI solutions to a free-of-charge, full-edged Web-based CAPI platform; for example, professional interviewers make intensive use of keyboard shortcuts and need adapted user interfaces. Also, the transition delay when leaving one question for the next should not exceed one second.

TAO was also enhanced to support highly complex flows of items where questionnaires make an extensive use of a) dynamic sequencing of questions (i.e., different answers lead to different follow-up questions), b) dynamic layout of questions (e.g., German words tend to be longer; white spaces are rare in Japanese; Asian characters need to be magnified by 10% to be readable; Hebrew and Arabic are read from right to left), and c) dynamic wording of questions (e.g., some language grammars require the wording of a question to take account of the respondent’s gender, age, and so on). The TAO toolbox integrated tools and processes allowing those dynamic aspects of the questionnaires to be tailored at the national level for each country participating in PIAAC.

Finally the number of partners involved in operationalizing the questionnaires and the diversity of the tasks to be achieved, often sequentially, sometimes concurrently, meant the need for well-defined organizational processes. These processes entailed the operations to create items, adapt

them to reflect national specifics, check their quality, push them to the final support (in PIAAC's case, a virtual machine), and so on. Therefore, the “process management extension,” based on an ontology-driven “workflow engine” (the first of its kind), had to be enhanced to handle a large quantity of active parallel processes.

In terms of software architecture, the workflow engine was not only responsible to support operationalizing questionnaires. The same workflow engine led the sequence of the noncognitive question items during the interviews. Initially defined by the specification document describing the PIAAC general workflow of the BQ, the sequence was adapted on a national basis.

The TAO architecture entails these **key features**, described in the sections below:

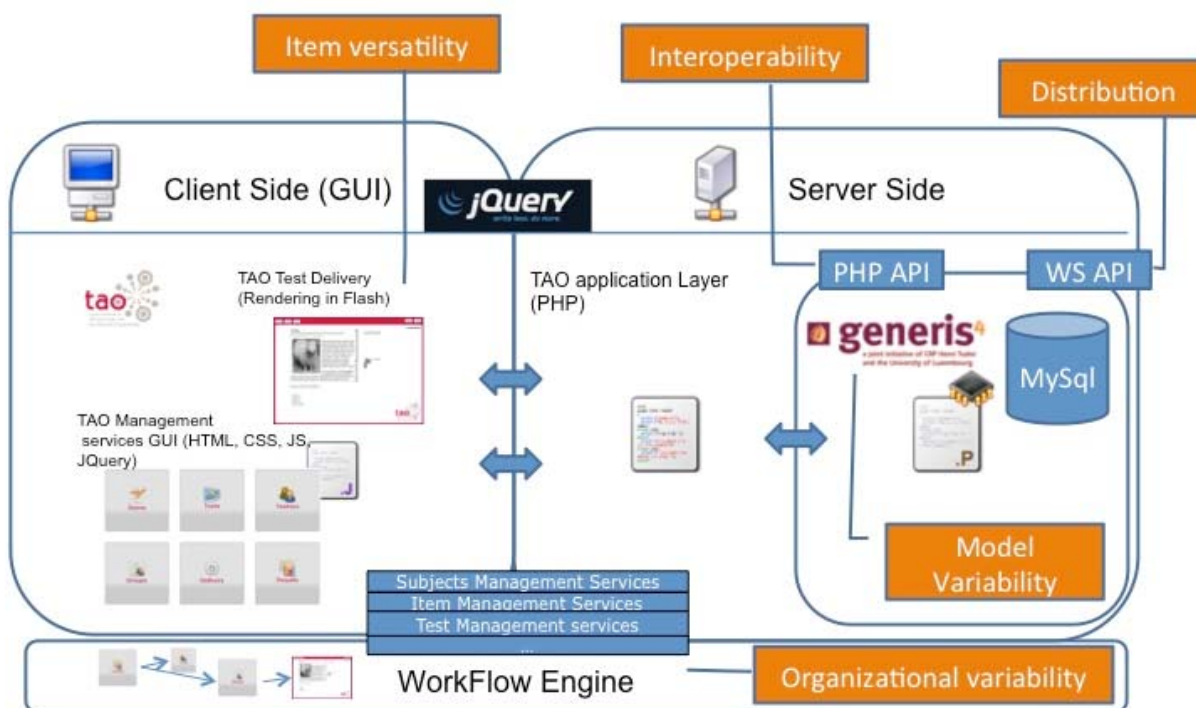
- Tackling technology-based assessment challenges with the semantic Web
- A workflow engine for cross-cultural large-scale assessment
- Test and item delivery architecture

9.2.1 Tackling CBA challenges with the semantic Web

Considering the range of variability of CBAs, a versatile architecture that makes use of specific innovative technologies was designed to tackle the needs of the different stakeholders.

Several types of variability needed to be addressed, not only in the context of assessment, but also to support the integration of CBA in e-learning environments (see Figure 9.2):

Figure 9.2: CBA challenges in terms of variability



Context variability: In the educational context, for instance, many different data models are created and managed such as the specification of the classrooms where the students sit, their teachers, their learning options, and so on. In addition, modern learning environments provide personalized learning situations that enforce the need to design assessment items accordingly. The annotation of the item could be used to personalize the assessment instrument, such as item selection or item layout adaptations that reflect, for example, disabilities or learning styles. These annotations could also be used to select more appropriate learning activities based on a student's performance on the different items. This model variability also applies to other resources in a CBA system such as subjects, tests, or management of the test results. From an IT perspective, this model variability is challenging because it prevents the definition of the data models a priori and the design of a classical database structure. To tackle this variability, we used the semantic Web-related technologies RDF and RDFS (Resource Description Framework Schema.) Both are standardized languages that enable us to express information about resources at any level of abstraction. They allow the system users to define the data model (i.e., to define classes of resources and describe their properties) as well as the data itself (e.g., to define values of properties that describe a particular student). All these data models are defined by using RDF/RDFS. They are created and adapted easily by the users of the system through intuitive user interfaces – no further implementation efforts are needed when the data model requires changes.

Using RDF repositories instead of a classical database design solves the model variability issue. The TAO platform makes use of the generis4 RDF/RDFS repository. This implies that, from the point of view of the application layer, the source code needs to be independent of the model, and all the user interfaces for resource management need to be generated by first inspecting the model that was defined by the user.

Interoperability: CBA systems need to be integrated into existing business processes and legacy software. This may also require replacing some existing feature subsets or extending the system with new features, which involves defining application programming interfaces (APIs) within the architecture that provide customized plug-ins to handle all the CBA resources (e.g., for computing statistics, creating proficiency reports, using existing subject databases, etc.). TAO provides two APIs, one for direct calls with the PHP programming language and one for remote access using SOAP Web services.

Distribution: CBA involves pedagogues, psychometricians, statisticians, item encoders and item translators, and may also involve stakeholders located in different geographic areas, such as a ministry of education guaranteeing access to subjects, a pedagogic institute defining tests, private software companies creating rich media for items, and so on. All the different stakeholders manage different kinds of resources. This requires all the CBA resources to be distributed on the CBA platform across a network of different collaborating institutions. Subject management, for example, is probably allocated to a specific accredited institution. Item management may be located at a different site to prevent items from being stolen. Such a distribution of tasks calls for the architecture of a CBA platform to be modular and distributable using the existing communication channels. This can be tackled through the use of Web services. Such distribution also allows sharing of resources and can be combined with a peer-to-peer network protocol, which would enable, for example, test creators to search across the entire network for items based on item model properties.

Organizational variability: The involvement of different stakeholders may require workflow-based work in order to make sure that the right person accesses the right feature of the CBA platform at the right moment. This also addresses the need for a quality layer to optimize the processes that lead to the creation of a measurement instrument. From an IT perspective, this involves the use of a workflow engine tool and a process design tool so that the person responsible for the assessment can design the CBA process according to his or her needs. This also requires that features from the CBA platform be split into autonomous services that can be triggered independently.

Item versatility: Authoring of items should not be restricted by the specificities of the CBA platform. It should allow simple item creation, like multiple-choice questions, but also more complex items, such as simulations. Maximum freedom should be given to the item developers at the level of the item layout and structure, as well as at the level of item behavior (interactivity). This can be achieved by a) defining a high-level language that supports layout, structure and behavior description, and b) implementing an authoring tool that facilitates the design of items as well as an interpreter capable of rendering such items.

9.2.2 A workflow engine for cross-cultural large-scale assessment

9.2.2.1 Introduction

With the advent of large-scale, cross-cultural surveys – such as IALS, Trends in International Mathematics and Science Study, Progress in International Reading Literacy Study, PISA and PIAAC – CBA becomes more strategic than ever before. While survey designers are used to facing adaptation and translation issues with paper and pencil, introducing the computer for such activities brings up new challenges.

Ideally, the very same version of a computer-based educational survey would be carried out in many different countries by simply translating its components. Unfortunately it might not be that simple. At an international level, cultural distinctness among areas most often leads to heavy modifications on how data collection in educational assessment will take place. From deleting questions to inserting brand new sequences of items midway through the assessment of some respondents, altering the original design of the material might be a frequent need. Depending on local constraints, participating countries will sometime have to modify how questions are asked and sequenced. Software instruments must also deal with a wide range of alphabets and symbols and need to be highly polymorphic to satisfy all stakeholders collaborating on cross-cultural large-scale surveys.

Deriving as many instruments as necessary to cover all countries' specific needs (lingual, cultural, socioeconomic, etc.) can be painful and error-prone work if not managed properly. To transpose a paper-based instrument into a computer-based one is also expensive in the context of a large-scale survey. To minimize the investments in time and money, a good approach would be “describe once, adapt as needed, and run many.” To reach this goal, we adopted a solution based on *workflows*, easing the production of extendable and executable assessment processes. By both adapting and running BQ and CBA built on the same computerized descriptions, we facilitated the design and implementation of cross-cultural computer-based educational assessment.

In the following sections, we describe how concepts and techniques from the world of workflow modeling and execution were applied to PIAAC. We introduce the formal representation of the assessment process based on the XML language. Then, we show how this format was used to support cultural adaptations. Finally we describe the execution of an assessment process in a computer-based context.

9.2.2.2 Describing the assessment process

Large-scale international computer-based surveys involve multidisciplinary teams from various countries across the world. This may lead to cultural and linguistic issues at the very beginning of the survey design; this adds up to the drawbacks that international projects usually encounter (e.g., agreeing on time-zone windows for meetings). The need for a formal representation about how data will be collected from an assessed population appears at an early stage of the engineering. A semantically sound pivot format giving stakeholders an opportunity to interact with a way marked assessment process description is essential.

According to Deelman, Gannon, Shields, and Taylor (2009), a workflow refers to defining the sequence of tasks needed to manage a business or computational science or engineering process. This also applies to BQ and CBA worlds by treating questionnaires and assessment sessions as sequences of tasks to be achieved by respondents. In this context, we say *assessment process* is the sequence of tasks required to assess and collect data for a single respondent. This statement is the basis for a simple but semantically rich format used to describe complex sequences of the tasks composing an assessment process and the rules that steer its flow.

A. Main concepts

Our XML description language integrates elements that authors can combine to describe processes including activities, transitions, consistency checks, variables and derivation rules. The following list explains the semantics of these concepts in detail and how they can be applied to the assessment world.

Process

It encapsulates the whole sequence of events occurring during the assessment process. It contains a set of activities logically linked by routing rules. It also holds the context of its process variables. The combination of the variable values and the current activity of a process instance constitute its state. In BQ and CBA contexts, a process represents an assessment session focusing on a single respondent.

Variables

They are comparable to simple data holders within a process. Our implementation gives every variable a global scope (i.e., variables are available within the entire process). Data collected during assessment will be stored in process variables.

Activity

It represents a task to be achieved during an assessment session. Activities are started according the flow logic of the process. They are divided into two categories: *Automated activities*, inherent to the process logic; do not need human interaction (Silver, 2005); they contain derivation rules or consistency checks. *Interactive activities* require human interaction and are focused on the respondent. For instance, such an activity could be an electronic reading item or a multiple-choice questionnaire.

Transition

It links activities using simple or complex logical rules. They might be used to direct the respondent to more or less difficult items during a single assessment process according to previously collected data, stored in variables.

Consistency check

It verifies the consistency of previously collected data. A violated consistency stops the assessment flow. Performing such checks at runtime is particularly useful for large assessment processes, including for the profiling of the respondent.

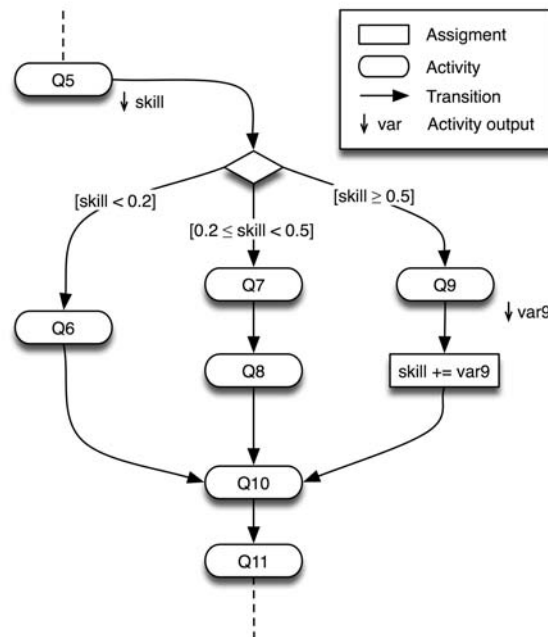
Derivation rules

Using derivation rules, authors can force the creation of variables at runtime or the assignation of a new value. Therefore new data can be generated based on values previously collected at assessment time. These derived variables will be used for subsequent derivations and flow control.

B. XML Description Language

Our XML syntax is used to depict assessment sessions. It is inspired by standards like BPML (Arkin, 2002) and XPDL that are currently developed and used by the industry sector.

Figure 9.3: A simple assessment process representation



Specially, the XML Process Definition Language (XPDL) specification offers a simple and minimal set of constructs present in most workflow products (van der Aalst, 2003). We used the following XPDL elements in addition to new ones to describe assessment processes (see Figure 9.3):

Workflow process

It contains the definition of activities, transitions and assignments.

Activity

It represents interactive steps and refers to the relevant software piece on which the respondent will interact.

Transition

It binds activities, transitions, assignments and consistency checks together using conditional expressions involving previously collected variable values.

Assignment

It contains logic setting a value to a particular variable by involving previously collected data. This is comparable to derivation rules but using XPDL semantics.

Consistency check

It verifies the consistency of the data contained in multiple variables.

Variable

It contains data collected during the assessment process. Transitions, assignments and consistency checks rely on their value to build up their logic.

9.2.2.3 Authoring and cultural adaptation support

To meet cultural requirements, stakeholders in large-scale and cross-cultural surveys need the right tools for cultural adaptations. They would like to add new cognitive items to the core survey in a particular sequence or suppress optional parts. Other partners may also need to add profiling questions related to the historical background of their country. All these adaptation needs are very difficult to foresee at the early stages of the assessment design. In this context, we developed appropriate tools to author, adapt and review XML-based assessment process descriptions.

A, Authoring companion tools

For the design and adaptation of assessment processes, we provided an authoring tool that did not require mastering the underlying XML format. Accessible via a Web-based user interface, it guides the author through the authoring process. Authors can comment on every modification they performed on the process description. This acts as a track-changes system, improving the communication between stakeholders. The authoring tool also prevented logical errors from occurring. For instance, a reference to an unknown variable in a flow transition was systematically detected and made visible.

Because activities, transitions and variable assignments may be numerous in a large assessment process, it was difficult to display all at once. To compensate for this shortcoming, the tool generates a graphical representation of the process based on the GraphML language (2010). This offered an easy way for authors to visualize the relationships between very distant elements of the process and to understand immediately its global design. This XML-based graphical representation is seamlessly produced using the XSLT technology (Clarck, 1999) aiming at transforming XML files using style sheets.

B. Continuous control

Creating numerous adapted versions of a reference assessment process (called *master*) must be carefully controlled. Critical aspects such as the comparability of collected data have to be taken into account. Thus, frequent reports were produced to set up a step-by-step reviewing process. These reports helped reviewers to validate or not the changes made by adapters. To support this, a

reporting tool able to detect differences between multiple versions of an assessment process description was created. To produce its reports, the reporting tool uses flat and hierarchical Diffing techniques (Miller & Myers, 1985; Chawathe, Rajaraman, Garcia-Molina, & Widom, 1996) in addition to the Levenshtein (1965) algorithm. As a result, additions, suppressions and modifications performed on an assessment process description are consistently underlined. Finally, the difference reports are delivered to reviewers in an XHTML format (Pemberton et al., 2000), usable on the computer but also easily printable.

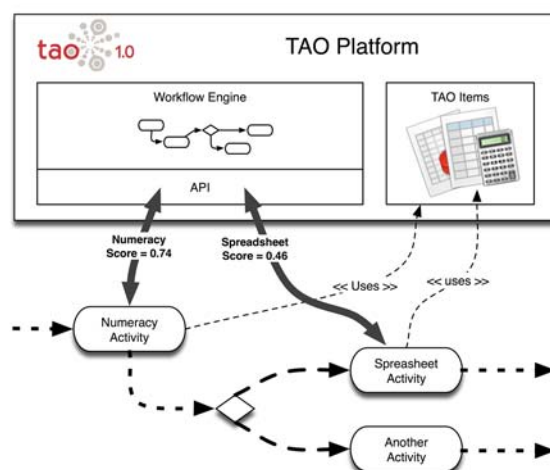
9.2.2.4 Assessment process execution

On top of our workflow description format, we built up a specific mono-user workflow engine (WE). It acts as a software service providing the runtime execution environment for a process instance focusing on the assessment of a single respondent at a time. Formerly described by authors, the assessment process is executed exactly as is. This helps to reduce the gap that may appear in computer-based projects between project designers and computer scientists.

A. The workflow engine

Our workflow engine is developed upon the PHP programming language (2010a). It was designed to be reusable as a Web or desktop (e.g., in a software built upon PHP-GTK) (2010b) application component as well. As a consequence, it can be embedded on both server and client sides. XML descriptions of the assessment process are processed and stored in the system, ready to be instantiated and run. As a test bed, we integrated our component into TAO: an open and versatile CBA platform (Latour & Martin, 2007, pp. 32-33). This enables users of this PHP platform to author workflows to drive simple to complex assessment processes with pre-existing TAO items as its interactive steps. As with any WE, it is in charge of instantiating assessment processes on demand. Each process represents the test session of a single respondent. These instances can, of course, be started, paused, resumed and destroyed at will. Each process has its own state, composed by its process variables values and its current activity. The engine takes care of sequencing activities in the correct order according to the process state and the routing rules it comes across (see Figure 9.4).

Figure 9.4: The workflow engine in the TAO platform



Activities depicting the tasks that have to be achieved at runtime are executed differently following their nature. A dedicated “rule engine” embedded in the software executes the logic bound to automated steps and evaluate routings. On the other hand, the execution of interactive steps is delegated to the appropriate software component. Currently, the engine only supports Web applications for “interactive steps” implementations. They are thus be provided to the respondent through a Web browser during the assessment process.

At runtime, rules attached in automated steps have an inherent access to process instance variables because of their technical proximity with the WE. By contrast, interactive steps are software parts developed apart from the engine. They need easy access to process contexts in order to influence the process flow if needed. To make this possible, the WE exposes an appropriate API. Through the latter, interactive steps implementations are able to consult and act on the process state by retrieving and modifying the value of its variables.

B. Assessment delivery

The assessment is delivered to the respondent through a Web browser. The graphical user interface (GUI) displayed to the respondent during an assessment session is the process browser (PB). It is in charge of proposing the correct interactive steps (e.g., cognitive items, questionnaires) to the respondent. The PB enables the user to interact with interactive steps implementations but also provide GUI components to navigate through the process. This allows the respondent to go back and forth between interactive steps and adjust previous answers if necessary. Each time the respondent decides to go forward through the assessment using the GUI, the WE gets the control back and selects the next interactive steps to be displayed.

9.2.2.5 Conclusion

We have reviewed the methods and techniques coming from the domain of computer-based workflows and how they were successfully applied to the PIAAC survey. The XML format and its semantics for describing assessment processes were satisfactory. Tools provided for the authoring and the cross-cultural adaptations of PIAAC processes were adequate. Finally, the workflow engine provided the expected support in that context.

9.2.3 Test and item delivery architecture

It is mandatory for tests and related items to be delivered the same way on any Web browser in order to prevent biases and discrimination. It is also important to be able to track the respondent’s behaviors (for CBA) or hesitations and latencies (for BQ) as those data may cast some light on subsequent results’ analysis. Moreover, to give a maximum of freedom to the authors for the design of rich items including media content is very desirable. Those constraints led to the choice of the Adobe Flash technology to create and deliver the test to the end user. The test and the item runtime engine read XML descriptions of the items that are authored beforehand with authoring tools. The XML-based items are based on the Business, Layout, Actions, Content, Knowledge Model (BLACK), which was developed at CRP Henri Tudor. The BLACK model is a high-level pivot format used to address the need for freedom required at the level of item creation. Thanks to the authoring tool, the user can edit items graphically and does not have to handle the item description file directly.

BLACK is composed of different sections. The *business* section of an item description gives overall information about the item. It describes those parts of the item that constitute the stimulus

(set of materials given to the subject, the part of the item that constitutes the task itself, the response categories). In addition, it also defines the right expected answer and the evaluation algorithm to be used. The *layout* section describes all graphical elements to be used in the item such as radio buttons, check boxes, images, and so on by using the XUL markup language (XML User Interface Language; 2010). The *action* section is used for items representing simulations, where certain elements of the item are expected to have a certain type of behavior. For instance, a pressed button might cause a Montgolfier picture to move up and down. The *content* section is language dependent and contains all text messages available in the item or all links to media, such as pictures or movies. Finally, the *knowledge* section contains metadata annotations of the item using XML RDF and describing potential skills, context of use, overall difficulty of the item, and so on.

9.2.4 Use of the TAO platform for PIAAC

Note: This section applies only to PIAAC Round 1. The BLACK format and eXULiS were not used for Round 2. See Chapter 5 of this report for more information.

9.2.4.1 BLACK model to support rich CBA items (component problem solving)

Motivations in using the BLACK format

The BLACK format addressed particular requirements of the PIAAC CBA platform to build and to run cognitive items, especially in the field of problem solving for technology-rich environments (PSTRE):

- Dynamically interoperate heterogeneous contents: e.g., for complex objects assembly;
- Facilitate contents extraction and maintainability: e.g., for localization (XLIFF);
- XML driven – based on namespaces to favor natural plugin construct and interaction;
- Encapsulation to assure packaging, transportability and deployment of the resources;
- Hierarchical nesting of contents from various data sources to favor interchangeability;
- Events-driven to assure the interoperability between the components;
- Homogeneous deployment on server side as well as on the client side.

BLACK — a MVC architecture

BLACK is an extension of the MVC pattern; it respects and elaborates on the MVC architecture principles. From a general point of view, the layers Business, Layout, Action, Content, and Knowledge can be successfully mapped to the Model, View, and Controller layers, as defined in Table 9.1.

Table 9.1: The mapping between BLACK and MVC

Model	View	Controller
Business		
	Layout	
		Action
Content		
Knowledge		

BLACK data structure

Streams of data structured according the BLACK format are named BLACK formatted streams or simply BLACK streams. The content of a BLACK data structure is a *Manifest* that references zero, one or many layers of the BLACK model.

```
<?xml version="1.0" encoding="UTF-8"?>
<black:manifest xmlns:black="http://www.exulis.lu/black.rdfs#" id="an_example">
  <black:business>
    ...
  </black:business>
  <black:layout>
    ...
  </black:layout>
  <black:action>
    ...
  </black:action>
  <black:content>
    ...
  </black:content>
  <black:knowledge>
    ...
  </black:knowledge>
</black:manifest>
```

This structure constitutes a normal basic BLACK stream; the root tag (i.e., “manifest”) must expose at least two attributes: a reference to the BLACK namespace and the “id” of the manifest. In the following section, we briefly present the definition of a manifest’s prime layers and show how this supported our goals and assertions regarding the authoring and rendering of PSTRE items described as BLACK bundles (i.e., one or more linked BLACK manifests and related resources (e.g., pictures).

The *Business* layer introduces the “semanticity” of all elements referenced by a manifest. Therefore, it operates as the driver of the integration and the interoperability of all the different kinds of contents involved in a bundle. It references the namespaces of the tags present in the BLACK stream and refers the components (e.g., parsers, services, etc.) to be loaded and invoked

dynamically when certain namespaces need a specific handling (e.g., for the rendering of a graphical element.) The layer also defines the bindings existing between elements of the *Layout* layer and data providers defined in the *Action* layer (e.g., services, functions) or existing as data container in the *Content* layer. This *Business* layer also holds all the preferences and settings subject to adaptation by the user and/or the system. It also contains directives for the correct handling of the BLACK stream itself; the directives indicates, among other things, where the BLACK stream should be handled: on the server side or on the client side. When a BLACK stream is processed on the server side, a specific parser, named *BLACKparser*, analyzes the stream and prepares a deployment infrastructure without any reference to the BLACK format so the final product results in a standard Web application where the use of BLACK becomes completely transparent to the server and the client. The BLACKparser is thus in charge to go from a meaningful compact BLACK manifest to a multi-tier application, and therefore, adding all the necessary in-between components enabling the expected execution of the whole application for the final user. A BLACK stream can also be carried along to the client side. In that case, the BLACK stream is handled by a *BLACKrunner* module. The BLACKrunner is available as two different flavors: one is based on the Adobe Flash technology and the other is built on a JavaScript technology. The Flash version of the BLACKrunner embarks the eXULiS library that natively handles the XUL and the SVG (scalable vector graphics) formats (even on browsers not compliant with these standards) and offers an extension mechanisms (to render other formats) and a toolbox facilitating rapid duplex communication, local storage, data processing, and enforcing resilience and security. The JavaScript version of the BLACKrunner is more dependent on the Web browser capabilities (e.g., the availability of the canvas object) but it clearly offers a broader compliance to the W3C standards on which it tries to capitalize a maximum (e.g., XHTML, CSS3, HTML5, and so on.)

The *Layout* layer is dedicated to the declaration of all the visual components displayed to the user (i.e., the side that was referenced as a part of the “Editor” in the first edition of the MVC (Reenskaug, 1979b) and was re-centered on the term of “View” in the second version of the MVC (Reenskaug, 1979a). These elements are usually related to a part or a complete framework offering graphical rendering capabilities; frameworks commonly available are XUL, SVG, XAML, XHTML+CSS, MXML, etc. Some of these frameworks are dedicated to specific domains as MathML, ChemML, and ChartML, in which they may need the support of an extra plugin in order to ensure the right display of the elements. The rendering of very advanced standards as the X3D and VRML also require the availability of specialized renderers installed on the client computers. When a BLACK stream is used as a support for a mashup aggregating various sources of contents, the Layout layer may summarize itself to a set of XHTML iframes (each pointing to its own contents) placed accordingly to a CSS. For this case, the added value of the BLACK is to enrich the default behaviors of each iframe by the injection of an *observer pattern* making the content of each iframe potentially reactive to the interactions occurring in other iframes (i.e., on other source of data.) An illustrating use case could be this one: Let’s suppose English is not your mother tongue. You connect to a portal offering a mashup specially designed to facilitate the reading of scientific texts available in English. After a light setup where you define your linguistic preferences, the reading facility would display two iframes; one is displaying a *Scientific American* article, and a second targeting *Google Translate* online application. Each time you would underline one word of the article, an event would be dispatched from this first frame; the event would be broadcasted by the BLACKrunner; an event listener attached to the second iframe would react and,

as a result, the application Google Translate would automatically display the translation of the highlighted word in your preferred language.

The *Action* layer is the container of all the logic processing. As the BLACK elements must be as loosely coupled as possible, all the interactions between the elements are assured by a proper events manager that is built on event dispatchers, event listeners, and if necessary event broadcasters. The logic (e.g., function, service, object method) registered to handle the catch of an event by a listener is naturally located in the Action layer. Also, all forms of processing, controllers, data access, and so on are nested in this layer. The BLACK format defines a limited set of elements that stands as a meta-language used to specify context of execution (i.e., server side or client side), loops, alternatives and control structures (e.g., “if/then/else”, “switch/case/default”), standard service calls, and so on. However, application logic needs more powerful capabilities. Therefore, it is possible to embark logic expressed in any language in a CDATA structure as long as it does not break the XML validity of the BLACK stream. Thus, because of its capacity to provide data via functional and service calls, the Action layer does not restrict to a pure Controller tier of the MVC architecture, but could also be considered as a part of the Model. Nevertheless, in our view, we prefer to assimilate all forms of processing, data manipulation, and service invocation present in the Action layer to the Controller part of the MVC architecture.

The *Content* layer is another part of the Model tier of the MVC architecture. Within the BLACK format it acts as a pure data container or as set of references to externalized data that themselves can be other BLACK streams. This may lead to extremely complex bundles made of imbricated BLACK manifests. In use cases where contents must be adapted depending on cultural constraints, it became totally obvious that all the data prone to localization and translation had to be located in this layer to facilitate their extraction by specialized processes (as met, for example, in international large-scale surveys) and their subsequent re-injection.

The *Knowledge* layer is a container for all the metadata describing the BLACK stream as a whole or parts of this stream (e.g., a description of a picture referenced in the Content layer, displayed in the Layout layer and voiced by a call – declared in the Action layer, to an external text-to-speech plugin announced, via the Business layer, as a capability to be initialized for the current BLACK stream.) This layer also specifies the format of the data to be produced as an output (if any) of the system described by the BLACK stream, for example, events to be collected.

9.2.4.2 eXULiS – a Rich Internet Application (RIA) framework used eTesting

Rich Internet Applications (RIAs) seem to be the solution offered by computer scientists to the more and more demanding users. Unlike previous generations of Web applications, RIAs plan the leverage of the user experience with more powerful GUIs including charts, drag and drop. RIAs also aim to remain as platform-independent and setup-free as possible. One should be able to work on data and tools both available online and these tools must be usable without setup or configuration on the client computer. However, RIAs also should be more than a few good-looking GUIs because the desktop applications have made the users accustomed to a high level of responsiveness and customizability.

RIA solutions are now legion. These are proposed to the developers as frameworks. Nearly none of these frameworks is really standardized even if the majority relies on the XML as a base for at least the GUI markup. The well-known application/UI formats are Mozilla’s XUL, Microsoft’s

XAML, Adobe's MXML, Laszlo Systems' LZX, ActionStep's ASML, ASWing's AXML (used for PISA 2009) and enFlash's ML.

For PSTRE, we used a new framework named eXULiS. It was not a standard or the ultimate RIA solution but rather another alternative that had been matured since 2006 and that tried to stay as close as possible to the standards of the moment. This section gives an overview of eXULiS and shows its place in the PIAAC assessment.

The overall architecture of the TAO platform

CRP Henri Tudor started to develop TAO in 2002. Until 2011, the platform relied on two main components. The first part, responsible for structuring, storing, and sharing of the data, was located on the server side. It was called Generis.⁴ The second part, active on the client side, was in charge of providing the correct display and completion of the tests and the questions (or items). It relied on the Flash player plugin that is installed on a large computer base (among individual computers). Since late 2011, more standard HTML5+JavaScript+CSS technologies replaced this latest component.

This section describes client-side technologies as used for PIAAC, prior to December 2011.

The work of the client-side component is to receive and to interpret the files describing the tests. The syntax of a test description is a mix of XUL and QML. XUL gives the layout of the graphical objects to appear on the screen of the computer for the test. We choose XUL for its maturity and because it was more open and community oriented than some other initiatives. QML stands for "Questions Markup Language." The QML used in the TAO platform was extended to encompass the needs of the IRT model as well as the new requirements towards the multilingual assessment.

The heart of the rendering engine is a parser called XUL2SWF (where SWF is the file extension of the Flash movies). The framework eXULiS is more than a simple evolution of the XUL2SWF engine. It contains a XUL parser completely rewritten to be extensible and more compliant with the Mozilla specifications. It also includes a second parser that is able to display SVG drawings. This XML language is used for describing two-dimensional vector graphics. The integration of a drawing format was required to open new vistas for the design of RIAs and for the authoring of advanced types of tests and items in the PIAAC context.

Authors who create new graphical layouts for RIA and/or for CBA can proceed using the tools freely available on the Web; XUL files can be written with xuledit.xul and SVG files can easily be produced with InkScape, for example. For PIAAC, we also developed a tool to adapt the XUL and SVG layout files. This tool was named Copernicus.

Figure 9.5 "TAO platform architecture" depicts the interactions of the two components described here above in order to deliver the RIAs, and more specifically, the tests to the test takers. This schema shows the topography (or deployment architecture) and some sequences of actions between the modules involved in the TAO platform. This schema should be read from left to right as everything in our platform starts from Generis (Plichart, Jadoul, Latour, & Vandenabeele, 2004).

On the server side, Generis provides, via its PHP API or its Web Services API, two sets of information issued from the RDF triples what it manages. The first of these sets of data can be used by third-party applications (authoring tools, eLearning platform, eBusiness applications, and

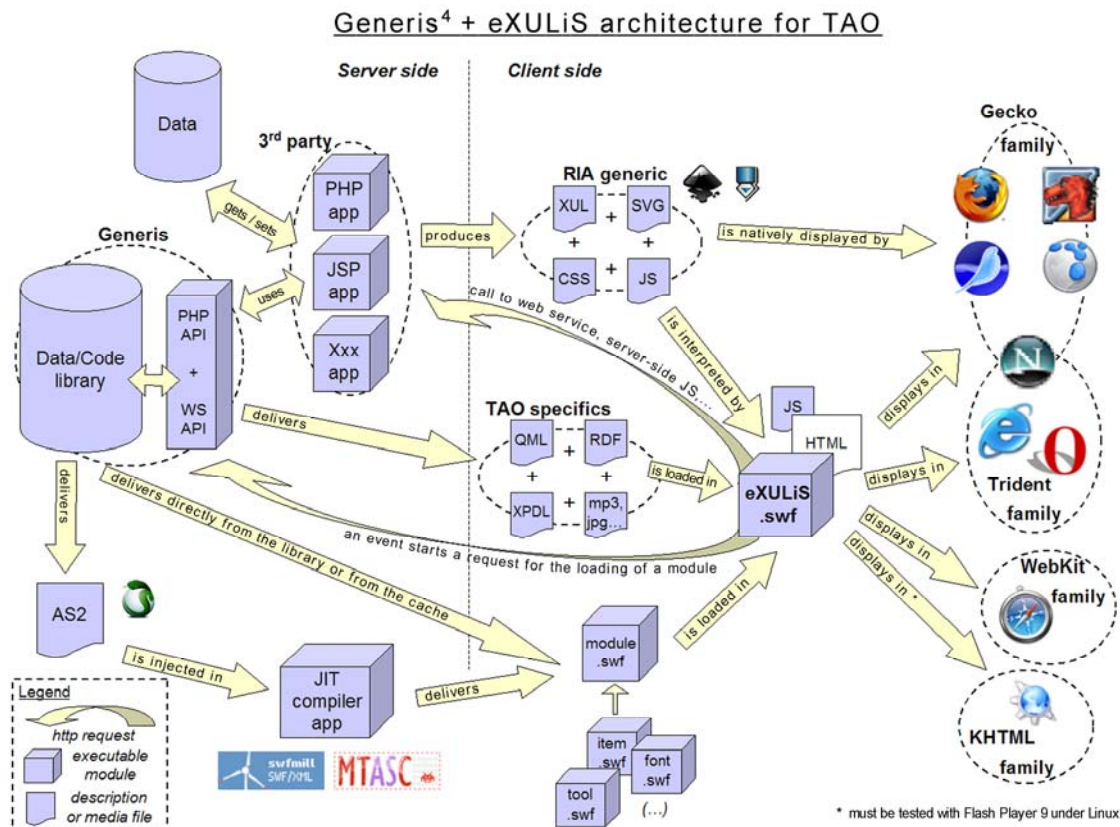
so on) to produce the RIA description files (XUL, SVG, CSS and JavaScript) that can be either natively rendered by the browsers embarking the Gecko Runtime Engine technology or that can be displayed in all the families of browsers via the eXULiS plugin. The second set of data contains the same kind of files as those involved in the first set but also includes files delivered in a format that is not directly interpreted by the Internet browsers (even the Gecko family). This part of the second data set is solely dedicated to eTesting. It holds some definitions that only have a meaning in the context of a CBA.

The files formats specifically used by eXULiS for CBA are the TAO QML definitions, the XPDL (XML Process Definition Language) and some specific XML and RDF datasets. Please note that, in the case of PIAAC, RDF information as well as XUL and SVG description were conveyed by the BLACK files described earlier in this document.

TAO QML files contain the logic and the hierarchical structure of the assessments. It means that the files describe a specific assessment in terms of a campaign involving one or several sequences of tests (potentially in different languages) including one or more sequences of Items made of a set of particular Items, each one integrating a Problem (stimulus), and Inquiries composed of a Question and a Distracter (e.g., a set of Proposals for multiple-choice questions, an open text, a puzzle, and so on). TAO QML is described later in this document

In the case of predefined sequences (called scenarii) of Tests, Items or Inquiries, eXULiS evaluates, at a moment T, the execution context of the assessment; then it uses some Workflow definition files formatted in XPDL that contains the conditions of the time T, to display the correct user interface at time T+1. As mentioned above, the definition of the GUI is not stored in XPDL but in XUL and SVG files.

Figure 9.5: TAO platform architecture



Another interesting aspect of eXULiS is its capacity to extend its dynamic behavior in numerous manners. The engine is a Flash movie (.swf file) and it can act as a relay to local or remote function calls. It embarks a set of wrappers and API that enables the invocation of Web Services, server-side JavaScript, local JavaScript (located in the Web page – HTML, PHP, JSP – that nests the eXULiS Flash object), and remote CGI scripts. It even allows the communication with client-side desktop applications through the use of local connections.

When it detects a specific need (for example, a Test event is raised requesting that the current Item is displayed in Japanese), eXULiS may forward this event to the Generis back office that will provide (if necessary via a Just In Time compilation) the useful resources (in this case a .swf module containing some Hiragana, Katakana and Kanji character sets). The .swf modules may contain diverse types of resources including fonts, tools (e.g., calculator, notepad), compressed XML datasets, media, and so on.

We will next briefly discuss the eXULiS and explain how its modular internal architecture is a favorable ground to extend its capabilities.

Design of the eXULiS framework

A two-fold construction

The effort to build the eXULiS framework started before the diverse initiatives led by the standardization agencies (e.g., W3C). Instead of creating a homemade GUI format for the specific

needs of the CBA, we decided to select one that was already available. We wanted an open standard, well established with enough resources available online, and if possible, a solid community; the Mozilla project called XUL, although not a standard, got selected.

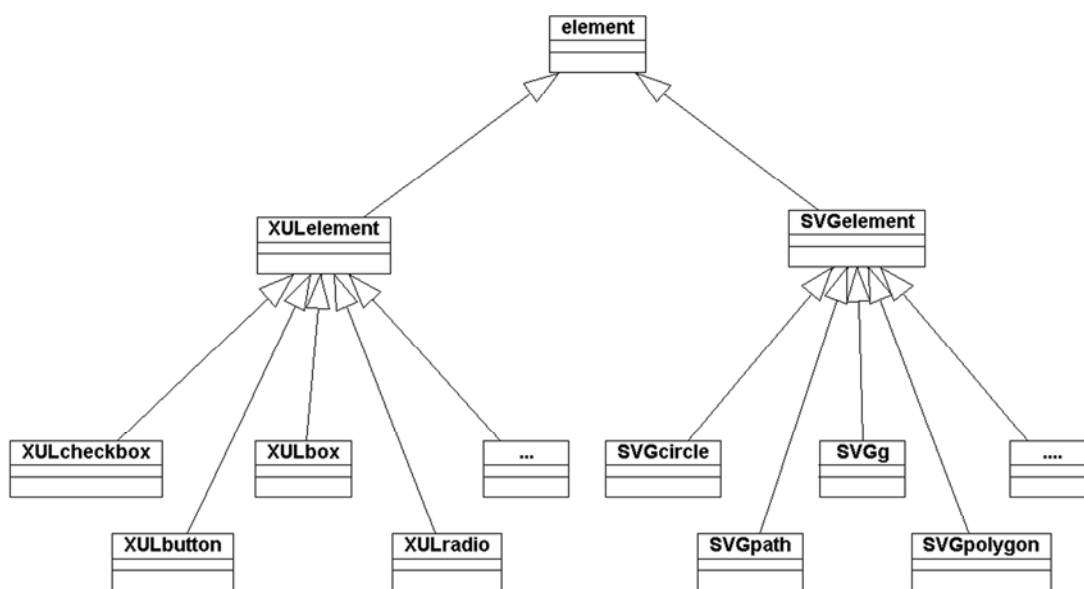
For our needs, we implemented this format in a component named XUL2SWF. Initially, this was a monolithic class encapsulating a parser that was recursively parsing XUL tags to render the corresponding GUI via the Adobe (formerly Macromedia) graphical widgets (e.g., basic “movie clips” and “v.2 components”). In 2007, about 60% of the XUL widgets were available in XUL2SWF, already enough to fulfill most of the needs of PIAAC.

In 2008, the one-piece class became the eXULiS framework and the coverage of the XUL specification increases as well as the new engine embarks other XUL-affiliated technologies like XBL (eXtensible Bindings Language) and RDF.

Furthermore, to address the new requirements elicited during the PIAAC survey preliminary analysis, we added a second framework to the XUL framework to handle the SVG standard.

In the Figure 9.6 “eXULiS framework overview,” the two parts of the class tree can be clearly identified: on the left side, XUL classes are inheriting from a common XULElement, and on the right side, SVG classes are inheriting from a common SVGelement; both XULElement and SVGelement are inheriting from element. The ancestor class element acts like a relay allowing a natural communication between the two frameworks, in particular via events. For example, widgets in the SVG framework can subscribe to any events of the type *xyz* and start to listen while a widget in the XUL framework can dispatch this type of events. The *xyz* event bubbles up to the root of the frameworks and gets broadcasted to subscribing SVG widgets.

Figure 9.6: eXULiS framework overview

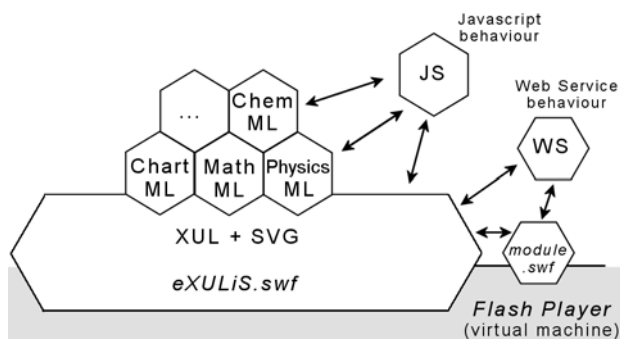


The whole framework can also use other static classes available in the Flash framework; an example is the Tween class that allows movements and transitions in the state of the widgets.

How can eXULiS be extended?

The integration of the SVG standard in eXULiS was first intended to allow authors to create their own custom themes and skins for the tests/items (e.g., a button with shape of a cloud for 6-year-old children). The power of the SVG standard and the capacity of Flash to call some external JavaScript functions unleashed the potentials of eXULiS. First, we created a module transforming some ChartML tags into SVG (see Figure 9.7) that eXULiS displays perfectly. Our latest projects target some needs in physics and genetics' laboratory simulations.

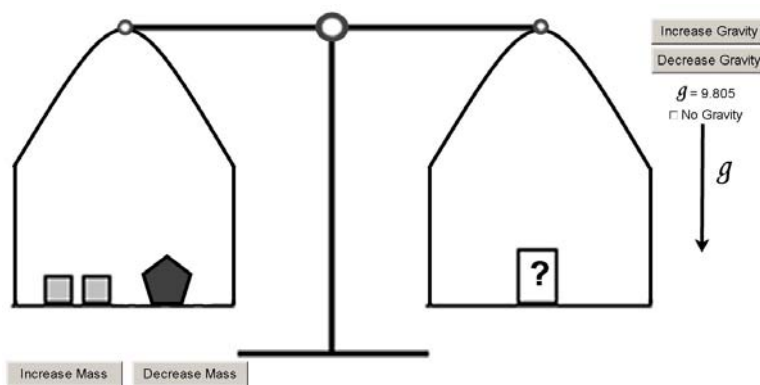
Figure 9.7: The eXULiS extension



An example of use of eXULiS

This example (Figure 9.8) is a mix of physics rules (gravity, levers, axis) applied to a schema that is a composition of SVG drawings and XUL widgets (buttons, checkbox) interacting in a laboratory allowing to experience a problem of static physics science.

Figure 9.8: A physics lab with eXULiS

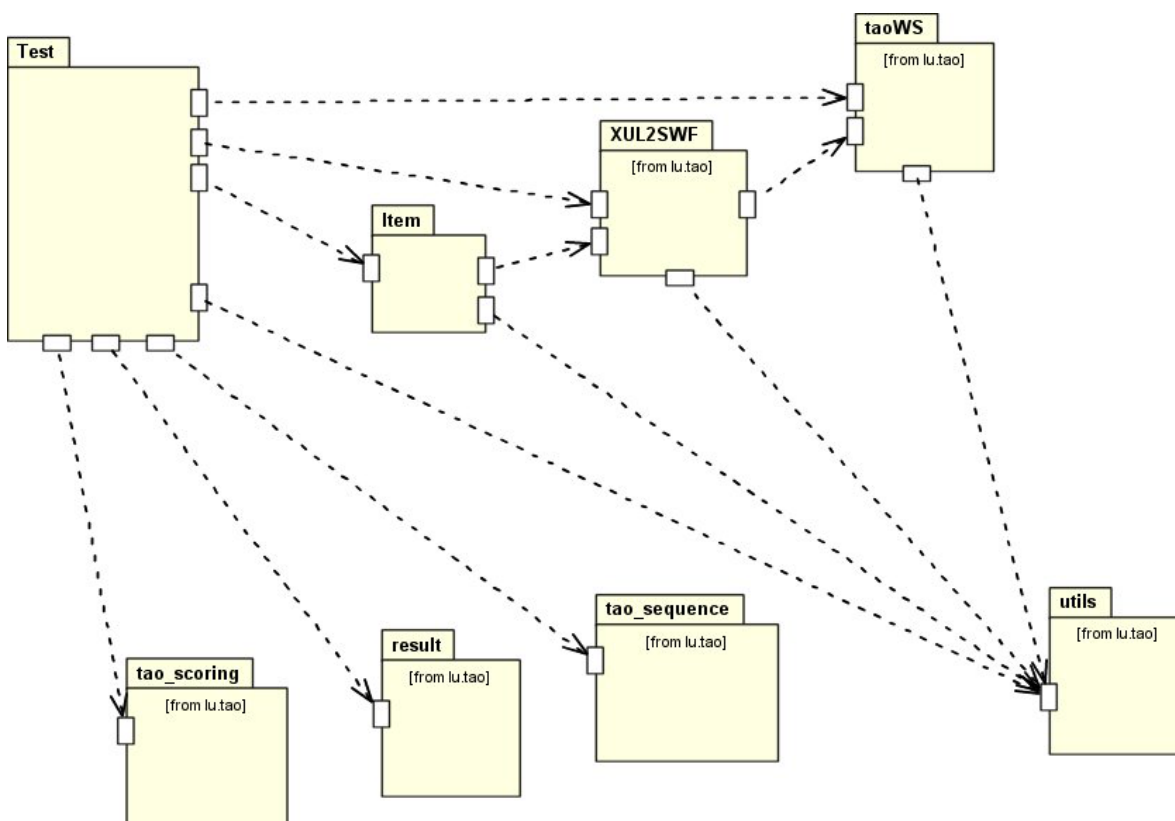


9.2.5 Test and item execution engine

The packages' organization of the Test and Item Execution Engine (TIEE) is centered on two main packages: the Test and the Item (cf. Figure 9.9). The following description of the Item package is based on the architecture of the QCM (Multiple-Choice Question) item model. However, the Item package structure may vary with the design choices taken by the item models designers (e.g., if designers use XUL2SWF/eXULiS to create PSTRE items). The only constraint for an item model is just to embark and connect its part of the communication interface in order to implement the

communication protocol that must take place between the Test instance and any Item instances. This protocol is described in the figures that follow.

Figure 9.9: TIEE packages organization



The Communication Interface (based on Flash Local Connection mechanism) between a Test and any kind of Item (see Figure 9.10) is defined in an API (see Figure 9.11).

Figure 9.10: TIEE packages organization

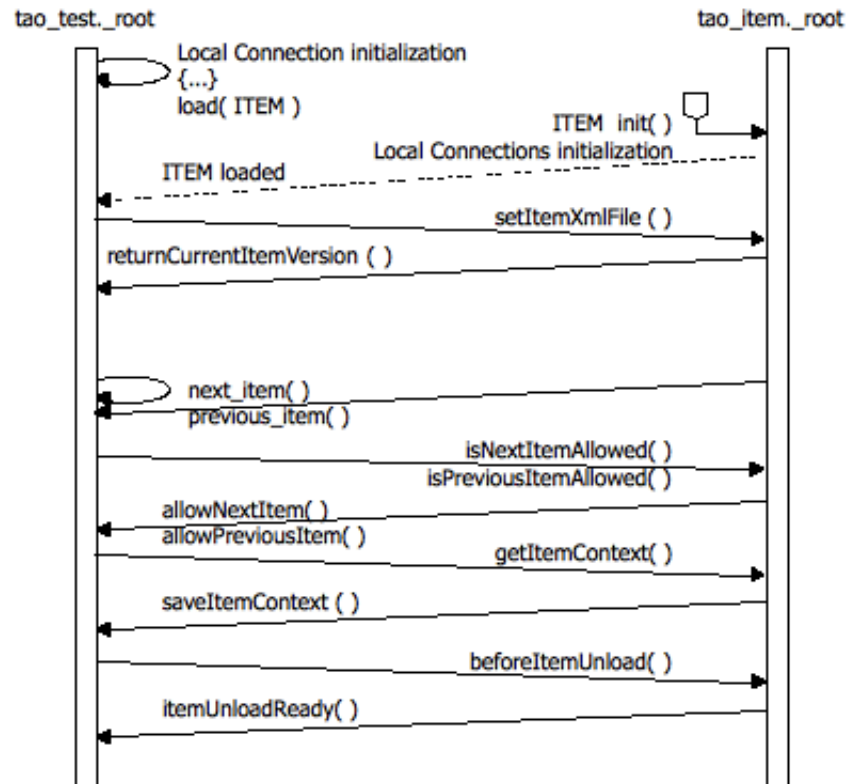
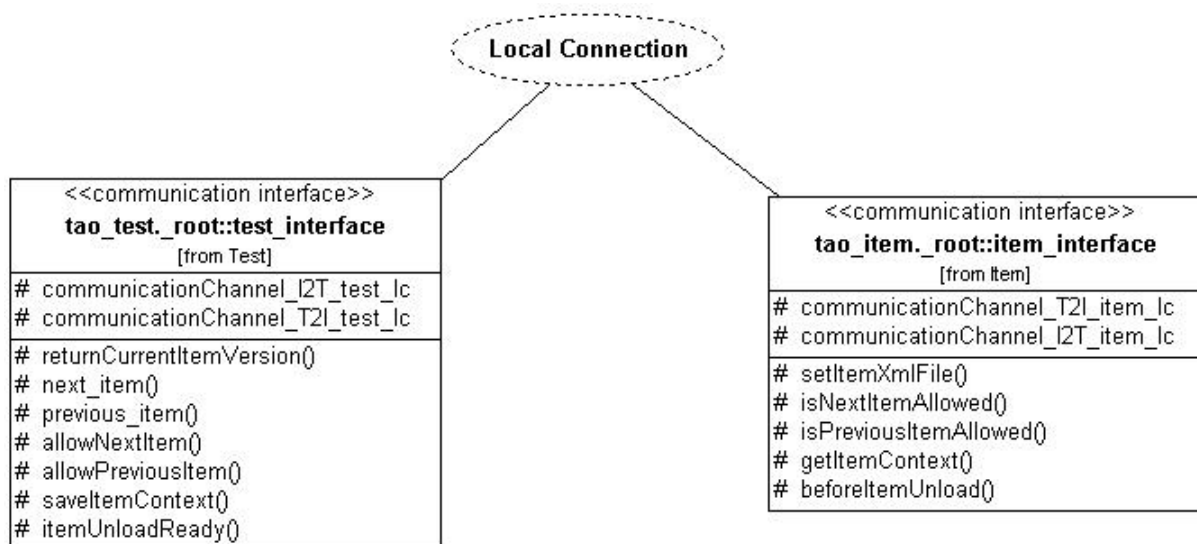


Figure 9.11: Communication API used in the Test and Item collaboration schema



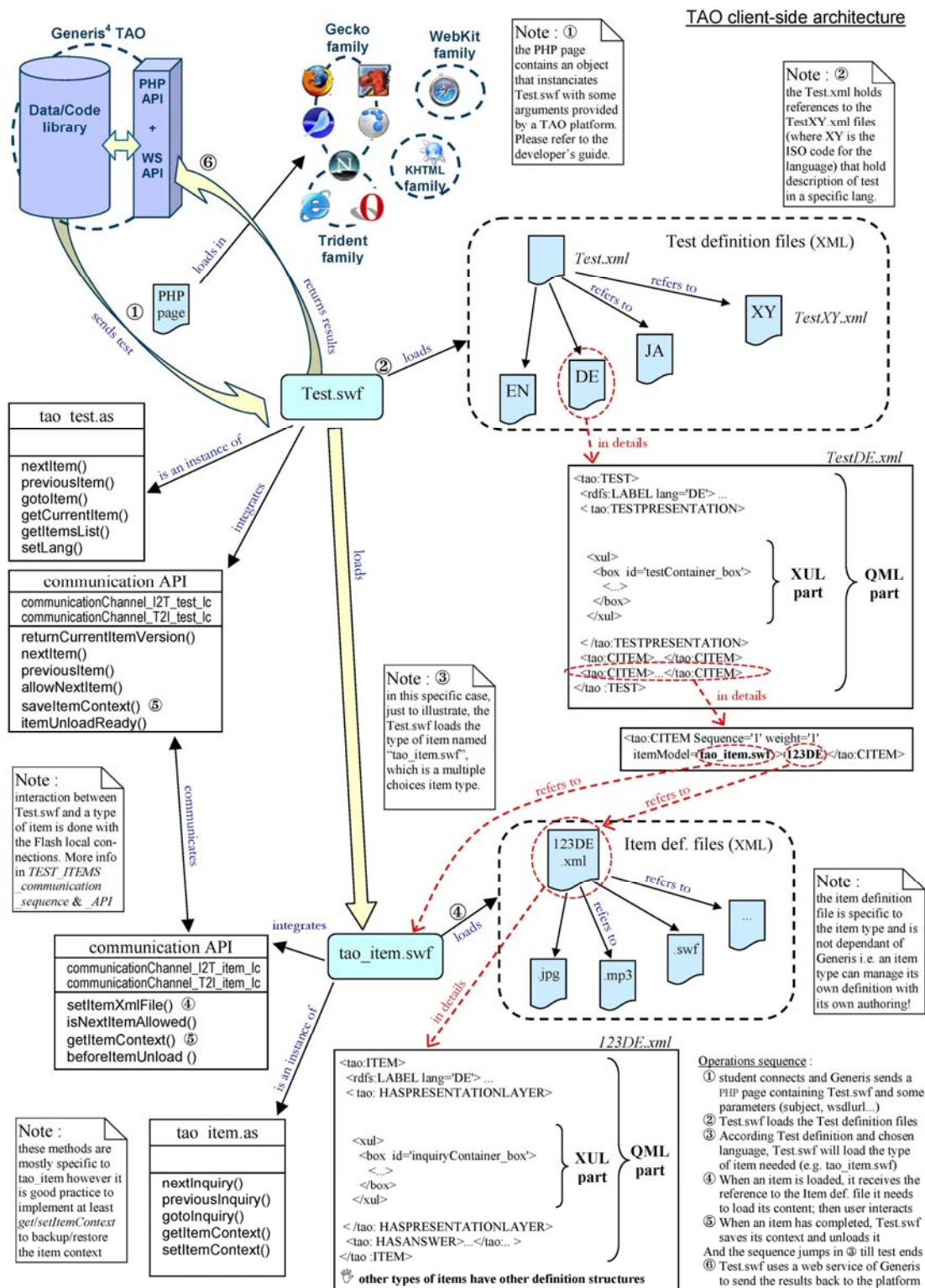
For the normal functioning of the TIEE, only the communication API (located in the item interface, on the right part of Figure 9.12) shall be integrated in an item model. Thereby, some item models already available on the TAO platform, present slightly different architectures (e.g., the text-with-

gaps item model). However, a good practice is to separate the control process from the graphical rendering. It is also good to avoid to split the Item's side of the communication API or to merge it here and there in the different parts of the Item architecture. For PIAAC, the developers applied Object-Oriented encapsulation paradigm, keeping the communication API in one layer of their item model movie clip, the GUI in a second layer, the data manipulation in a third one, and finally the control and process actions in one or more other layers.

In the case of PIAAC, although the Test was responsible for the activation of the scoring and items sequencing, the design of the TIEE was oriented to a delocalization of the scoring and sequencing algorithms in separated packages. This architectural choice ensured a minimum maintenance cost while new scoring (largely dependent on cultural specificities; for example, the comma separator and the thousand separator vary from country to country; in Japan, even a ten-thousand grouping the numbers is usual) and sequencing methods were adapted or added to the TIEE.

Figure 9.12 shows the standard execution and data flows of the testing process activities as managed by the TIEE. In the schema, at the level of step 4, the structure of the item XML definition file (here, 123DE.xml) varied depending on the design choices made by the developer of the item model; the model used in the example is the QCM that uses the tao_item.swf Flash execution file.

Figure 9.12: Schematic illustration of the Test and Item execution



References

- Arkin, A. (2002). *Business process modeling language*. Retrieved from xml.coverpages.org/BPML-2002.pdf
- Chawathe, S. S., Rajaraman, A., Garcia-Molina, H., & Widom, J. (1996). Change detection in hierarchically structured information. In J. Widom (Ed.), *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* (Montreal, Quebec, 4-6 June, 1996), 493-504. doi: <http://doi.acm.org/10.1145/233269.233366>
- Clark J. (1999). XSL Transformations (XSLT), Version 1.0. Retrieved from <http://www.w3.org/TR/1999/RECxslt-19991116>
- Deelman, E., Gannon, D., Shields, M., & Taylor, I. (2009). Workflows and e-science: An overview of workflow system features and capabilities. *Future generation computer systems*, Vol. 25, 528-540.
- GraphML Team, The. (2010). *GraphML specification*. Retrieved from <http://graphml.graphdrawing.org/specification.html>
- Latour, T., & Martin, R. (2007). TAO, an open and versatile computer-based assessment platform based on semantic web technology. *ERCIM News*, 71, 32-33.
- Levenshtein, V. (1965). Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR* (Russian), 163(4): 845-848. English translation in *Soviet Physics Doklady*, 10(8), 707-710.
- Miller, W., & Myers, E. W. (1985). A file comparison program. *Software – Practice and Experience*, 15(11), 1025-1040.
- Pemberton, S., et al. (2000). XHTML™ 1.0 The Extensible HyperText Markup Language, A reformulation of HTML 4 in XML 1.0 (2nd ed.). Retrieved from <http://www.w3.org/TR/2002/REC-xhtml1-20020801>
- PHP Group, The. (2010a). *Introduction to PHP*. Retrieved from <http://www.php.net/manual/en/introduction.php>
- PHP Group, The. (2010b). *PHP-GTK: PHP-GTK 2 Manual: Preface*. Retrieved from <http://gtk.php.net/manual/en/preface.php>
- Plichart, P., Jadoul, R., Latour, T., & Vandenabeele, L. (2004). Communication at the advances in intelligent systems – theory and application. *AISTA 2004*. Luxembourg.
- Reenskaug, T. (1979a). *Models - Views - Controllers. Technical note*. Xerox PARC. Retrieved from <http://heim.ifi.uio.no/~trygver/mvc/index.html>
- Reenskaug, T. (1979b). *Thing-Model-View-Editor - An example from a planning system. Technical note*. Xerox PARC. Retrieved from <http://heim.ifi.uio.no/~trygver/mvc/index.html>
- Silver B. (2005). *BPMS watch: Agility and BPMS architecture*. Retrieved from <http://www.bpm institute.org/articles/article/article/bpms-watch-agilityand-bpms-architecture.html>
- van der Aalst, W. M. P. (2003). *Patterns and XPD L: A critical evaluation of the XML process definition language*. Eindhoven, Netherlands: Department of Technology Management, Eindhoven University of Technology.

XML User Interface Language (XUL) (2010). Retrieved from
<https://developer.mozilla.org/en/XU>