Generating User Interfaces from CDISC ODM for Mobile Devices

Guido M. de Melo, Jürgen Nagler-Ihlein, Michael Weber
Dept. of Media Informatics, University of Ulm, Germany
{guido.de-melo, juergen.nagler-ihlein, michael.weber}@uni-ulm.de

Abstract

Clinical studies are often conducted as multi-centered studies involving participants at different locations. Thus it becomes obvious that using mobile platforms and remote data entry is beneficial for such studies. When working in a distributed fashion, data exchange standards are required. Bringing these standards to the end user also requires incorporating user interface issues. Considering various device types, from desktop computer to small handheld devices raises the question how user interfaces could be derived from the data exchange standards. In this paper we consider CDISC´s Operational Data Model as a data standard and discuss its relevance in the user interface generation process. Several candidate description languages are explored and embedded into a general transformation process emphasizing especially small and mobile device characteristics.

1. Introduction

Traditionally clinical studies are conducted by filling in forms on sheets of paper and later analyzing the data gained. Due to widespread use of computers it becomes obvious, that the data handling as well as data entry can be supported by such devices. However, since a personal computer, even a laptop, is not really mobile and cannot be carried and used at the same time on-site during a study, initial data entry still remained mostly on paper and data entering was repeatedly done afterwards on the computer again. This raises problems of inconsistencies through mistyping, with the entire process being time- and cost-intensive. On-site electronic data capture (EDC) has been proposed to alleviate this problem. Further advantages of EDC systems are the legibility of the entered data and the possibility to use range and plausibility checks while entering the data, thus further improving data quality.

Over time a diversity of electronic clinical study systems has emerged, all of which use their own digital formats for storing and conveying data. This makes it difficult to exchange data during a study between the involved parties and systems. The required conversion or adaptation process is time-consuming, prone to errors, and also cost-intensive. Hence it is important to use specific standards when developing new tools in order to make them usable for a broad community. Fortunately standards for study data have emerged recently.

Information technology is embedded into more and more devices, while people have come to depend on digital services. At the same time, there is a growing diversity of devices and their range is expanding in modalities as well as in platforms: mobile phones, handhelds, and notebooks to name only a few.

These mobile devices can easily be given to patients, study nurses, or doctors to be carried on-site, which makes it easier to conduct studies.

However, such remote data entry (RDE) is still fraught with some problems. Mobile devices have smaller displays and no keyboard or they are not connected to a network. Many mobile devices cannot run the software used for entering the data, since their operating systems and APIs are too different from the desktop computers being used so far. At the same time they offer other input modalities like pen-based input, which is in turn not supported by the traditional software used for studies. It is possible to write an application which can run on any platform, but adaptation to the special features of each device is another matter, as code written to run everywhere cannot be optimized for every platform and also gets bloated.

The challenge is therefore how to utilize a standard description of case report forms (CRFs) to support remote data entry on heterogeneous mobile devices.

The reminder of this paper is organized as follows. In section 2 we describe the operational data model (ODM) as a standard to define CRFs. Section 3 deals with the generation process of deriving a user interface (UI) from such models. In section 4 requirements on user interface descriptions to mediate in the generation

process are outlined. Section 5 evaluates various candidates to serve as a user interface description language. Our findings are presented in section 6.

2. CDISC ODM

The Clinical Data Interchange Standards Consortium (CDISC, www.cdisc.org) is an open, multidisciplinary, non-profit organization founded in 1997. Its mission is to develop platform-independent global data standards for clinical research. The standards should assure information system inter-operability and lead to improvements for medical research and related areas of healthcare. CDISC's first success was to get its Study Data Tabulation Model (SDTM) accepted by the FDA as a standard for data submissions of clinical trials. While the SDTM was developed to improve the exchange of results after a trial, another CDISC standard, the Operational Data Model (ODM), was developed to support all data processes during a trial and to be a common format for archiving study data.

With all the advantages of EDC in view, the FDA encouraged the CDISC electronic Source Data Interchange (eSDI) Group [1]. The group intends to facilitate the use of electronic technology by establishing CIDSC standards within clinical trials, particularly the ODM. This is a strong indication for ODM becoming a standard for electronically representing a study, and it makes ODM a good choice of a starting point of our approach.

2.1. Design of ODM

The ODM [2] is defined in XML and for its current version 1.2.1 the XML Schema is considered definitive. It is able to keep study metadata, study data, and administrative data of a clinical trial. As the intended use of ODM is to support the exchange and archiving of trial data, only the information which needs to be shared among different systems and to be stored for regulatory purposes is part of the model. Therefore, parts like the graphical representation of CRFs or a concrete structuring of input fields are neglected.

The ODM is divided into four big parts: <Study> containing the structural definitions (meta-data) of the study; <ReferenceData> for general data not concerning a specific study subject; <ClinicalData> storing the clinical values for each study subject; and <AdminData> keeping information about users, locations, and electronic signatures.

The most interesting part of ODM with regard to the user interface of a trial is its Study section. It contains global settings like the study name, a description of the study, and measurement units. As keeping all changes to the study metadata is supported, the versioning of the study is realized by different MetaDataVersions. The current state of the study is the aggregate of subsequent MetaDataVersions. Within a MetaDataVersion all input fields are defined within structures in a strict hierarchical manner. The highest definition levels are StudyEvents. Each StudyEvent defines one or more Forms. A Form then contains ItemGroups containing Items. These are the smallest entities and correspond directly to an input field.

An Item is described by its name, data type, data size, and a question used to label it on paper or on a screen. Possible data types are integer, float, date, datetime, time, and text. For a float the SignificantDigits have to be specified. There are a number of options to characterise the Item further. One (the default) or more (all valid) MeasurementUnits can be defined, or an optional CodeList which lists a discrete set of permitted values. With RangeChecks one is able to formulate comparisons as one-side constraints with one or more membership checks towards a set of values.

As clinical data systems frequently store more information than can be expressed by the given ODM elements, a vendor extension mechanism is provided for proprietary extensions. This gives the possibility to compliantly add and transport additional data which may be needed by an application working with the core ODM data.

2.2. Insufficiencies of ODM

While working with ODM we discovered that some data types are missing as well as some points in which the ODM should be extended to further support the conduct of a study. Some of these aspects are being discussed by the ODM team and may in the near future be incorporated into the standard. To preliminarily overcome these obstacles we employ the vendor extension mechanism described above. In case the standard would not be extended, our adaptations can be kept as vendor extensions without interfering with the rest of ODM data.

In our view, forms contain more data types than the set provided by ODM; especially Booleans and arrays of values are often used in trials. It is of course possible to store all conceivable data as text but then no semantic information on the data is available. Such semantic information can be used to provide adequate input elements and we therefore introduce an

additional attribute SubDataType for Items which carries this information.

RangeChecks, the possibility of validating input against its data type, and the declaration of mandatory Items are a first step in the direction of achieving better quality by checking data while capturing it. Immediate checks of input values can give the user an instant hint or mark values as invalid. So in the monitoring process fewer errors are found, and fewer queries have to be sent to and processed by the investigator. But a broader variety of checks, syntactic and semantic, can be imagined, e.g. complex syntactic checks, dependencies between input fields, or calculations of differences in date. We have therefore developed a vendor extension for checks within Items.

Many parts of CRFs have always been left blank as they are only needed if a specific precondition is met. A famous example is the question of pregnancy in dependence of gender. Such fields are conditional and their condition can be formulated like the checks for validity. As not only Items but whole StudyEvents, Forms, and ItemGroups can occur conditionally, we built a vendor extension element Condition which can include checks that can even be applied to these elements.

The structuring mechanism of ODM for Forms only allows having two levels: Items within ItemGroups. So Items belonging together as regards content should be within one ItemGroup. But with our conditional structures in mind we see the need for a stronger structuring mechanism. If some of these Items depend on the same condition it would be wise to put them in a subgroup.

As stated in the ODM specification event scheduling and time ordering of StudyEvents, Forms, ItemGroups, and Items are not part of the specification so far. Such data would provide additional value, e.g. appointments could be generated. To merely generate user interfaces, the included attribute OrderNumber for ordering sub elements is sufficient.

3. Generation of User Interfaces

To solve the problem of mapping one application to many platforms, we propose a model-driven architecture (MDA) approach [3]. For our purposes such an approach is applied to the mapping of ODM towards a user interface. Generally, in an MDA approach an abstract platform-independent model is transformed into a platform-specific model from which, in our case, the final UI can easily be derived automatically.

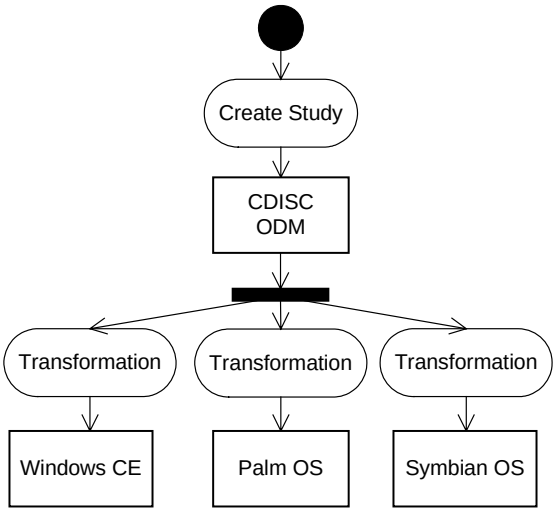

Figure 1: Transformation of CDISC ODM to different platforms.

3.1. Conceivable Approaches

To generate a user interface a platform-independent model is needed. There are basically two options, an executable tailored to a specific platform can be generated or the model can be rendered directly on a device. Figure 1 shows the steps involved from the creation of a study to deploying it on the target devices. ODM takes the place of the platform-independent model here.

3.2. Generating Platform Specific Code

If an executable is desired, a transformation has to be written for every device which needs to be supported. Such a transformation contains a mapping from the abstract model to concrete interface elements as well as adaptations which are platform specific.

To implement this approach, platform-specific transformations need to be designed and implemented. By employing a cross compiler chain, executables can be produced for each target platform. This would be a costly approach, however, since all the transformations would have to be written as well as adapted to the various mobile devices.

3.3. Interpreting ODM

In case ODM should be interpreted on the device itself, facilities for parsing the model and generation of the user interface have to be provided. The transformation then takes place on the device.

This variant is applicable for platforms with a higher computing capacity like notebooks, whereas the former variant (section 3.2) is preferred for devices which cannot provide many resources. Parsers and generators have to be written for every target platform. The generators should make use of platform-specific methods to make the most of the user interface.

3.4. Employing a User Interface Description Language

Since both of the above approaches necessitate a huge effort, a third approach seems more feasible. There are existing user interface description languages (UIDLs), which support and ease both approaches described above. By transforming ODM into a suitable UIDL the transformations or interpretations are taken care of and only a minor transformation needs to be written.

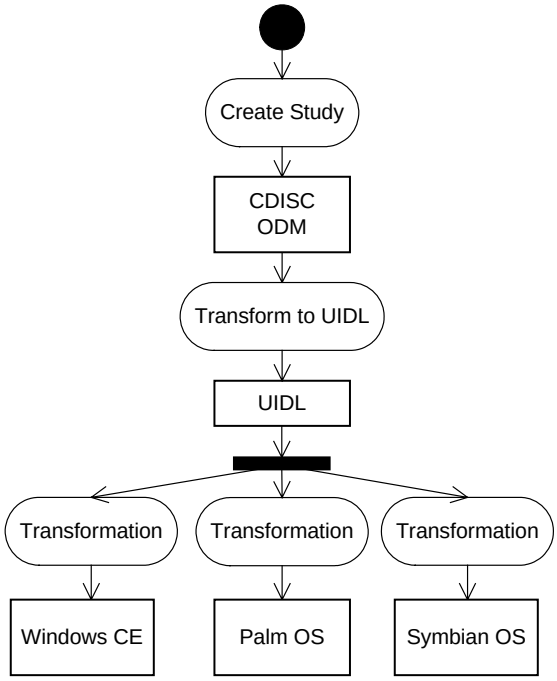

Figure 2: Transformation of CDISC ODM to different platforms utilizing an intermediary UIDL.

The transformation from ODM to a UIDL will benefit from the fact, that the envisioned UIDL is also an XML application. Therefore, a huge range of tools is readily available to support this transformation step. As figure 2 shows, only one specific transformation needs to be facilitated, while the transformation steps towards the device platforms can be utilized from other projects dealing with generally transforming from a UIDL to a concrete user interface.

4. Requirements for UIDLs

To get a study environment running on different mobile platforms out of one description, a lot of requirements have to be met at different levels. The ODM has to be extended to keep all necessary information concerning a good semantic description of the study in the face of operating the study with mobile devices.

4.1. User Interface Aspects

As far as the interface is concerned there are primarily two aspects to be considered.

First, how completely can the CRF be mapped from ODM onto a concrete user interface? The more elements are available for that purpose the better. It is also important how elements which are not transferable can be described by combinations of different available elements.

Second, how good is the support for mobile devices? CRFs should be adapted to small screens and a large number of small devices should be supported as well. This process is ideally taken care of automatically.

The possibility of limiting specific adaptations to a platform is also important.

4.2. Programming Language Aspects

Concerning languages to be employed for describing the interface there is the aspect of the behaviour of the system at run-time for interpreted languages which tend to be rather slow since it poses a heavy strain on the device's resources. In the case of compiled languages no performance penalties are to be expected.

For both compiled and interpreted languages there remains the aspect of their expressivity and power as an important criterion in selecting a language.

A language employed to describe a user interface for CRFs should support mappings to many different platforms as well as a wide variety of widgets and interaction patterns. At the same time it has to strive to be as generic as possible. Mappings to new platforms should be possible without too many adjustments to the mapping itself.

Such a language should also provide for a way to represent the sequence of interactions in a CRF. This

should be mapped to a flow between the widgets of the concrete interface.

Hierarchical grouping of form elements should be possible as well as inactivating conditional subgroups in case they become irrelevant because of data entered before.

On every supported platform it is expected that the semantics of the CRF can be made good use of, e.g. a pen-based device should toggle its input mode to numbers or text according to the field currently being edited.

After data entry, it should be possible to perform syntactic checks on the data entered and to provide the user with feedback on mistakes, such as data which are out of a range.

5. Evaluation of UI Description Languages

By applying the above criteria to different UIDLs it is possible to evaluate them and decide on the language suited best to the objectives.

5.1. XForms

XForms is a technical recommendation developed by the World Wide Web consortium [4]. Its focus is to provide enhanced platform-independent forms for web-browsers where the layout is taken care of by the browser. Of course, an XForms-enabled browser has to be available for all target platforms. Integration into an existing (web-based) framework for RDE is simple, since only the forms have to be distributed. Validation of entered data can be done by employing constraints in the XPath language.

Semantic support for entering data depends largely on the integration of the browser into the platform. As XForms is an interpreted language, it will in all likelihood not be feasible on small devices like palmtops or mobiles, at least for complex CRFs. However, XML4Pharma [5] is a company employing the XForms approach.

The elements of the ODM Study section are mapped onto XForms elements. Validity checks are mapped onto XPath and XML Schema Definition expressions.

5.2. UIML

UIML was developed to facilitate building device-independent user interfaces while promoting the separation of the interface from the application logic [6].

A large number of technologies for small mobile devices is supported, e.g. Java/JFC, PalmOS, WML, HTML, and VoiceXML. The interface is adapted to small screen resolutions automatically. Since it is possible to interpret UIML at run-time and to generate binaries for a number of target platforms, performance problems are not prone to arise.

Automatic validation of entered data is not provided per se, but can be contributed during the generation process. Semantic support for data entry is also possible.

The elements of the Study section of ODM are mapped onto abstract user interface elements. Validity checks would have to be translated into a programming language which in turn incapacitates the platform independence.

5.3. XIML

XIML was introduced as a UIDL to support design, operation, organization, and evaluation functions in the UI creation process. It provides abstract as well as concrete elements and employs a multi-tier architecture.

XIML is designed to support a wide variety of devices and thus to support adaptation to small displays. The concept leaves open the question of whether binaries are being generated or the language is being interpreted. Semantic support for data entry would be available if supported by the target platform [7].

The Study elements of ODM descriptions are transformed into abstract user interface elements. XIML is still in a stage in which it is impossible to anticipate how translation of validation checks is to be achieved.

5.4. TERESA

The Transformation Environment for inteRactivE Systems representAtion, TERESA in short, is an authoring-tool [8]. An abstract task model is created in the form of a *ConcurTaskTree* [9], of which the user interface is generated in multiple steps. TERESA is multi-platform approach featuring adaptable automation. By defining relations between multiple abstract representations and abstract user interfaces it is possible to describe the dynamic behavior of a system.

The utilization of an abstract model as well as the multi-platform functionality favor this approach. Since the final user interfaces are generated for each target platform, performance should be good.

Support for mobile devices is not complete; what is more, before any automatic generation is possible, transformations have to be adapted to each platform

because of the different interaction patterns. Integration into a web-based framework might be possible, but it would not prove simple since the complete tool-chain has to be adapted to the framework.

ODM descriptions need to be converted into *ConcurTaskTrees* to be used, which is possible in principle. However, in order to automate the UI generation further research is needed since it is unclear how the semantic information contained in ODM could be used to an advantage in generating the concrete interfaces.

6. Conclusions

ODM offers a good solution as a description language for studies including CRFs. However it was not designed to be used as a description language for the corresponding user interfaces as well. Therefore, we propose that by employing vendor extensions and transforming ODM into an intermediate UIDL a mapping to mobile platforms offers many gains.

ODM has certainly developed to be the language of choice for describing CRFs as its standardization shows. However, transforming ODM directly through a rendering process to use it on many platforms is very costly as has been shown above. Employing an intermediate UIDL relieves the developer of the task of implementing all the relevant transformations himself, thus offering a sound solution to the problem.

There are several ways to implement a solution which have been outlined in the section above. Although a high-level tool like TERESA seems to fit our approach best, we decided differently. The transformation of ODM into *ConcurTaskTrees* is complicated and the ensuing processing of the *ConcurTaskTrees* necessitates too much manual work.

Transforming ODM to UIML is straightforward, since most of the descriptions contained in ODM can be mapped to descriptions in UIML. The available UIML transformation then takes care of adapting the descriptions to the mobile platforms being employed.

We are currently working on a mapping from ODM to UIML to facilitate a prototype implementation. Our goal are UIML descriptions which address special features and pitfalls of our target platforms. The resulting interfaces will be generated for and tested on a range of small mobile devices which were selected on a basis that they would fit typical clinical scenarios. As a first result an implementation using UIML.net [10] is currently under test. In the near future this implementation will enter user trials to evaluate the results.

7. References

[1] CDISC eSource Data Interchange (eSDI) Group: "Leveraging the CDISC Standards to Facilitate the use of Electronic Source Data within Clinical Trials", http://www.cdisc.org/eSDI/forms/eSDI.pdf, Version 0.5, 2005

[2] CDISC ODM http://www.cdisc.org/models/odm/v1.2.1/index.html

[3] Jishnu Mukerji and Joaquin Miller, “Overview and guide to OMG's architecture”, Object Management Group, http://www.omg.org/mda, 2003.

[4] http://www.w3.org/MarkUp/Forms/

[5] http://www.xml4pharma.com/XForms/index.html

[6] Constantinos Phanouriou, "Uiml: A device-independent user interface markup language", dissertation, Blacksburg, Virginia, U.S.A., 2000.

[7] Angel Puerta, and Jacob Eisenstein, “XIML: a common representation for interaction data”, *Proceedings of the 7th international conference on Intelligent user interfaces*, 2002, pp. 214-215.

[8] Silvia Berti, Giulio Mori, Fabio Paternò and Carmen Santoro, “TERESA: A transformation-based environment for designing multi-device interactive applications”, *Proceedings of the 9th international conference on Intelligent user interface*, 2004, pp. 352-353.

[9] Fabio Paternò, Cristiano Mancini and Silvia Meniconi, “ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models”, *INTERACT*, 1997, pp. 362-369.

[10] Kris Luyten and Karin Coninx, “Uiml.net: an Open Uiml Renderer for the .Net Framework”, *CADUI’2004*, pp. 260-273, 2004