OPEN ACCESS

ccML, a new mark-up language to improve ISO/EN 13606-based electronic health record extracts practical edition

Ricardo Sánchez-de-Madariaga,[1] Adolfo Muñoz,[1] Jesús Cáceres,[1] Roberto Somolinos,[2] Mario Pascual,[1] Ignacio Martínez,[3] Carlos H Salvador,[1] José Luis Monteagudo[1]

► An additional supplementary appendix is published online only. To view this file please visit the journal online (http://dx.doi.org/10.1136/amiajnl-2011-000722).

[1]Telemedicine and Information Society Department, Health Institute 'Carlos III' (ISCIII), Madrid, Spain
[2]Bioengineering and Telemedicine Laboratory, University Hospital 'Puerta de Hierro' Majadahonda, Madrid, Spain
[3]Communication Technologies Group (GTC), Aragón Institute for Engineering Research (I3A), University of Zaragoza (UZ), Zaragoza, Spain

Corresponding author
Dr Ricardo Sánchez-de-Madariaga, Telemedicine and Information Society Department, Health Institute 'Carlos III' (ISCIII), Avda. Monforte de Lemos 5, 28029 Madrid, Spain
ricardo.sanchez@isciii.es

Received 1 December 2011
Accepted 1 September 2012
Published Online First
27 September 2012

ABSTRACT

Objective The objective of this paper is to introduce a new language called ccML, designed to provide convenient pragmatic information to applications using the ISO/EN13606 reference model (RM), such as electronic health record (EHR) extracts editors. EHR extracts are presently built using the syntactic and semantic information provided in the RM and constrained by archetypes. The ccML extra information enables the automation of the medico-legal context information edition, which is over 70% of the total in an extract, without modifying the RM information.

Materials and Methods ccML is defined using a W3C XML schema file. Valid ccML files complement the RM with additional pragmatics information. The ccML language grammar is defined using formal language theory as a single-type tree grammar. The new language is tested using an EHR extracts editor application as proof-of-concept system.

Results Seven ccML PVCodes (predefined value codes) are introduced in this grammar to cope with different realistic EHR edition situations. These seven PVCodes have different interpretation strategies, from direct look up in the ccML file itself, to more complex searches in archetypes or system precomputation.

Discussion The possibility to declare generic types in ccML gives rise to ambiguity during interpretation. The criterion used to overcome ambiguity is that specificity should prevail over generality. The opposite would make the individual specific element declarations useless.

Conclusion A new mark-up language ccML is introduced that opens up the possibility of providing applications using the ISO/EN13606 RM with the necessary pragmatics information to be practical and realistic.

INTRODUCTION

For more than a decade, the need for semantically interoperable electronic healthcare records (EHR) has become evident.[1–4] Current healthcare provision would be unrecognizable without the organizational principle of integrated care that is enabled by EHR. However, their semantic interoperability is a desideratum that is still far from being reached. Until very recently EHR systems failed to implement the so-called dual model that separates knowledge from information. Instead, they used a single model in which the domain concepts were hard-wired into the software. As a result, the systems that evolve in environments characterized by a high degree of complexity and speed of change of the concepts that are managed soon become obsolete if they are not updated, in a process that turns out to be too frequent and expensive.[4] In recent years considerable efforts have been made in order to spread the dual model and overcome these problems. Nevertheless, at the moment transparent EHR transmission between healthcare providers is not yet an everyday reality.

Several organizations such as ISO,[5] CEN,[6] HL7,[7] openEHR[8] and recently the CIMI initiative (clinical information modeling initiative)[9] have been working towards the achievement of semantic interoperability of systems using EHR. The ISO/CEN 13606 standard is based on a dual model in which the concepts at the knowledge level are modeled semantically by archetypes;[10–14] it has been used successfully in the development of several systems.[15–18]

This means that EHR extracts (an extract is the clinical record of a patient or a part of it: it is the EHR communication unit) are constructed using the building blocks provided by the ISO/EN 13606 and ISO 21090 reference models (RM), and their semantics is refined by the constraints imposed by the ISO/EN 13606 archetypes to it. This framework can be used to define in precise and unambiguous terms an EHR extract in the form of an XML archive. This model is accurate and exhaustive, ie, it provides all the syntactic and semantic information needed to transfer the information conserving its whole context. However, its completeness also forces it to introduce a lot of medico-legal and implementation context information. (In order to make things simpler, in the remainder of this paper we will refer to these data as context information.) This means that for instance an average extract includes a very high proportion of such data, very often obligatorily. A small statistical study performed on a sample of 50 EHR extracts showed an average 74.5% proportion of this information (see supplementary appendix, available online only). This makes the extracts compilation by the healthcare professional impractical, because he or she must manually introduce a lot of context information data. A mechanism that would enable that information to be filled out automatically in each particular case would indeed facilitate EHR extracts creation.

These considerations lead to the necessity of creating an additional external mechanism that complements the RM without modifying it, providing the pragmatic information needed to enable the healthcare professional (or a user of any other application using the RM, or even a computer

Open Access
Scan to access more
free content

system) to generate EHR extracts in practice, ie, preventing him or her from introducing endless context data and facilitating concentration on the essential medical aspects.

This is a feature with broad applicability in a number of use cases within a variety of practical applications with highly structured information, such as automatic documentation generation (bibliography, laws), scientific knowledge systematization (botanic, pharmacology, etc.); in fact, it could be considered an entire efficient software development paradigm.

In order to meet these goals a new XML-based mark-up language has been designed and tested: the ccML language, for 'context completion mark-up language'.

BACKGROUND AND SIGNIFICANCE

As outlined in the introduction, one of the aims of the ISO/EN 13606 standard is to normalize the transfer of information between EHR systems in a semantically interoperable way, so that the recipient system of the EHR extract is able to understand it as if no change of location or context had taken place. Several current paradigms in the creation of standards were followed by the CEN technical committee 251—health informatics. These include separation of responsibilities, separation of points of view and separation of information and knowledge.[4]

The last mentioned paradigm constitutes a novelty in the design strategy of the standard and is responsible for semantic interoperability. This is achieved by means of a dual model approach that gives rise to two well-differentiated kinds of concepts: a small set of generic concepts (RM), the grammar of the domain, managed by the developer; and a large number of domain concepts (called archetypes), understood and described by their specialists. In this case, if the knowledge level suffers any modifications, they remain at this level and do not affect the implementation of the system on the generic concepts of the RM.[10–14]

The ISO/EN 13606 data structures represented in the W3C XML schemas implementing the RM include a very important amount of mandatory data to be filled in the extract. This information ranges from technical meanings of concepts or reports introduced to coded identifiers of persons, data or coded intervals of the time assigned to many issues. The use itself of archetypes facilitates this task because part of the information generated to be included in the extract is stored in them. Nevertheless, further help is required in order to simplify this process as much as possible.

As an example of the convenience of supplying the editor with additional pragmatic information consider figure 1. It illustrates the fragment of the RM's XML schema where element *ehrExtract::EHRExtract* is declared; several situations related to the use of context information follow:

EHRExtract uses two obligatory elements (*rmId* and *timeCreated*) among others.

Element *rmId* has a *fixed* value, 'EN 13606', which is stored in the schema, and thus could be fetched by the system from there. Element *timeCreated* has type *dateTime*, which indicates that its value is the current date and time at the creation of the extract, but this value could be computed by the system more accurately and efficiently than by the user.

Element *ehrExtract::EHRExtract* has also four different successor elements with type *II* (*instance identifier*, defined in ISO 21090) three of which are obligatory. In figure 2 the declaration of type *II* in the ISO 21090 schema is shown. It also uses attributes inherited from its base classes *ANY* and *HXIT* also depicted in the figure.

The first attribute *root* of type *II* has a system-wide given value for all extracts created by it and can be predefined a priori. This value can be given to the system before interpretation in a single place, and all elements of type II can copy it from there. In our system this attribute has the value '2.16.724.4.21', which is the OID given by the Spanish health ministry to the Madrid regional government. The second attribute extension from the same type must have a system-wide unique string value. A different specific value is computed for each instance of this attribute. Attributes *validTimeLow* and *validTimeHigh* represent the time margins of validity of the identifier. In our system these intervals start at the moment of creation of the object and have no end point. So it is computed by the system at object creation; attribute *validTimeHigh* has as a predefined value (PV) representing a very distant date in the future as 1 January 3000 (the string '30000101').

Many of these elements are discussed further later. At the moment note that when the user wants to create an *ehrExtract::EHRExtract*, the root element of every new ISO/EN 13606 extract, the system would require him or her to type (and sometimes precompute) the content of 14 fields all of which are context data and could be fetched or precomputed by the system automatically. Even though there may be relatively simple ways to preconfigure the completion of these fields in a system that uses a RM like this one for whatever purpose, in this research we introduce a new tool, the ccML language, that is generic enough to allow the use of any application-independent RM as long as this RM is compliant with the dual model archetypes framework. Moreover, the user can write a file defining how the system should behave in these situations; this definition can be performed in real time and does not require applications to be modified. For instance,

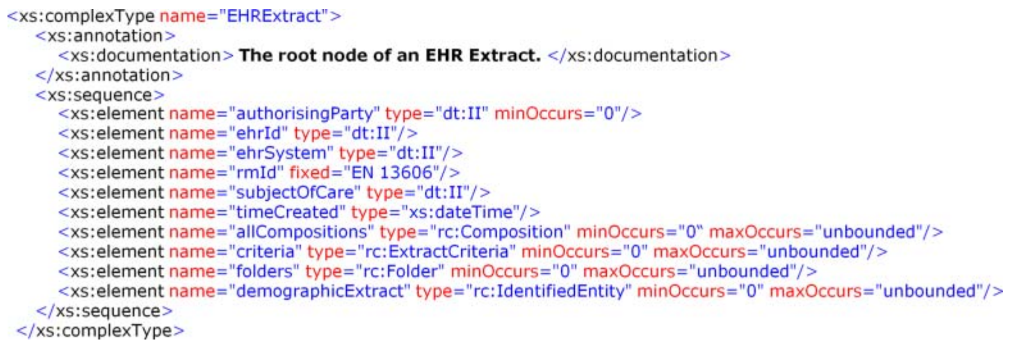
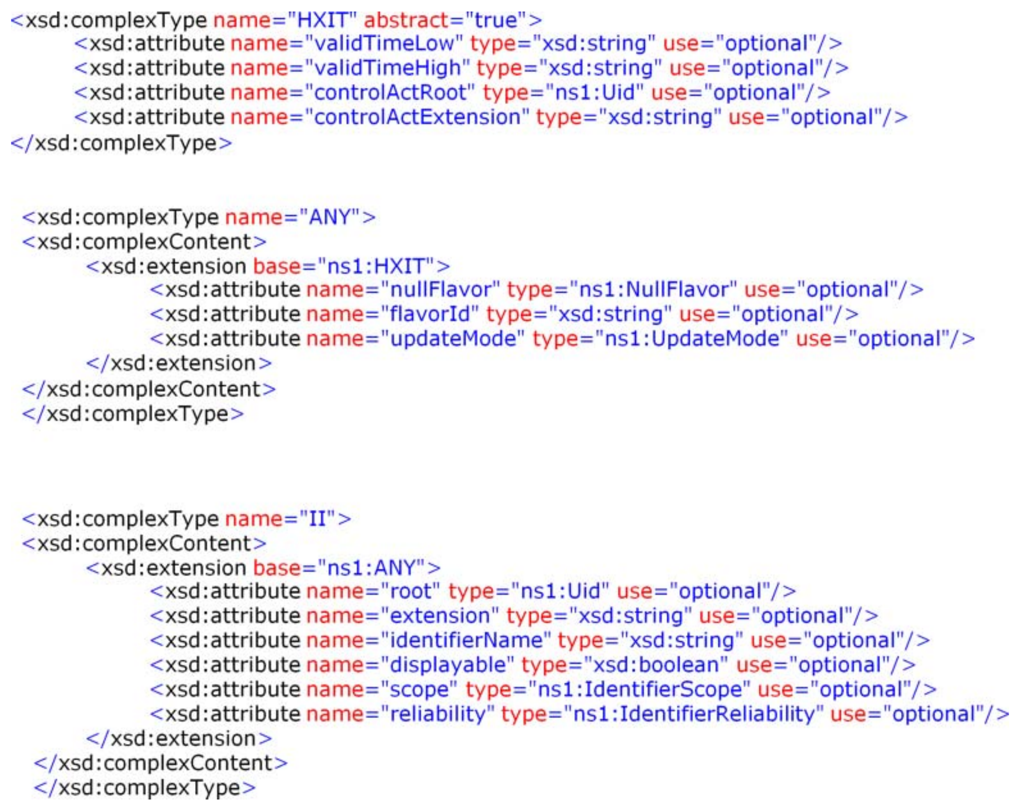

```
<xs:complexType name="EHRExtract">
  <xs:annotation>
    <xs:documentation> The root node of an EHR Extract. </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="authorisingParty" type="dt:II" minOccurs="0"/>
    <xs:element name="ehrId" type="dt:II"/>
    <xs:element name="ehrSystem" type="dt:II"/>
    <xs:element name="rmId" fixed="EN 13606"/>
    <xs:element name="subjectOfCare" type="dt:II"/>
    <xs:element name="timeCreated" type="xs:dateTime"/>
    <xs:element name="allCompositions" type="rc:Composition" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="criteria" type="rc:ExtractCriteria" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="folders" type="rc:Folder" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="demographicExtract" type="rc:IdentifiedEntity" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

Figure 1 A fragment of the XML schema defining the ISO/EN 13606 part of the reference model (extracted from Sun, Austin, and Kalra, in *ISO EN 13606 part 1 XML form V.3*, University College London, 2011, personal communication).[11] This figure is only reproduced in colour in the online version.



```
<xsd:complexType name="HXIT" abstract="true">
      <xsd:attribute name="validTimeLow" type="xsd:string" use="optional"/>
      <xsd:attribute name="validTimeHigh" type="xsd:string" use="optional"/>
      <xsd:attribute name="controlActRoot" type="ns1:Uid" use="optional"/>
      <xsd:attribute name="controlActExtension" type="xsd:string" use="optional"/>
</xsd:complexType>


 <xsd:complexType name="ANY">
 <xsd:complexContent>
       <xsd:extension base="ns1:HXIT">
             <xsd:attribute name="nullFlavor" type="ns1:NullFlavor" use="optional"/>
             <xsd:attribute name="flavorId" type="xsd:string" use="optional"/>
             <xsd:attribute name="updateMode" type="ns1:UpdateMode" use="optional"/>
       </xsd:extension>
 </xsd:complexContent>
 </xsd:complexType>


 <xsd:complexType name="II">
 <xsd:complexContent>
       <xsd:extension base="ns1:ANY">
             <xsd:attribute name="root" type="ns1:Uid" use="optional"/>
             <xsd:attribute name="extension" type="xsd:string" use="optional"/>
             <xsd:attribute name="identifierName" type="xsd:string" use="optional"/>
             <xsd:attribute name="displayable" type="xsd:boolean" use="optional"/>
             <xsd:attribute name="scope" type="ns1:IdentifierScope" use="optional"/>
             <xsd:attribute name="reliability" type="ns1:IdentifierReliability" use="optional"/>
       </xsd:extension>
  </xsd:complexContent>
  </xsd:complexType>
```

Figure 2 A fragment of the XML schema defining the ISO21090 part of the reference model. Extracted from ISO 21090:2011 health informatics.[14] This figure is only reproduced in colour in the online version.

following the previous example, if the system were used in a different region, such as Castilla La Mancha, with a different OID the new value could be stored in the ccML file in real time and the application could continue as is.

As stated above, according to our experience, approximately over 70% of the information could be generated this way. Consequently, a new tool that allows the definition of how the system should fill those data automatically would be very useful.

MATERIALS AND METHODS

The ccML language

In order to overcome this problem we propose as a solution the use of a new language that enables the system to generate the context data automatically.

In figure 3A the classical architecture of an extracts editor application is shown. It accepts as input RM XML schemas and archetype ADL files,[19] and produces as output an XML file representing a new extract that complies with the input RM and the archetypes restrictions.

The ccML mark-up language introduced in this paper is defined in its own XML schema, and .xml extension files can be written in ccML.

XML is a meta-language for creating mark-up languages.[20] To create one such language from XML, a collection of names for elements and attributes used by the language are created. This is specified by an XML schema language, W3C XML schema, in the case of this work.[21]

ccML grammar analysis using formal language theory

Murata *et al*[20] use regular tree grammar or tree automata theory to capture schema languages formally and document validation against those schemas.[22 23] Traditional context-free or regular grammars are designed to describe permissible strings in programming languages or natural languages; they are inappropriate for describing permissible trees, which is the structure of XML files. On the other hand, regular tree grammars are designed for that purpose.

The grammar of ccML as a single-type tree grammar is shown in figure 4.

Single-type tree grammars are a special case of regular tree grammars in which competition of non-terminals is prohibited. This means roughly that two different non-terminals (inside set N in figure 4) A and B cannot stand in the left-hand sides of two different productions (set P in that figure) that share a non-terminal in their right-hand sides. Notice from Murata *et al*[20] that W3C XML schemas (on which ccML is defined) correspond to single-type tree grammars.

Different ccML files can be created by the user of the editor in order to solve specific data generation problems and needs. For instance, if the editor had to be used somewhere other than Madrid, the ccML file could be modified so that attribute *root* from the example above would receive a different OID value.

To specify how the different components should behave, ccML allows the definition of:

- Data type (see production *PVType* in figure 4): allows defining restrictions for every instance of a data type (from ISO 21090).
- Complex elements (production *PVComplexElement)*: to define restrictions on a complex element of the RM (composed of other complex elements, simple elements and attributes).
- Simple elements (production *PVSimpleElement*): to define restrictions on a simple element of the RM (it can only have a value and attributes).
- Attributes (production *PVAttribute*): to define restrictions on an attribute of the RM.

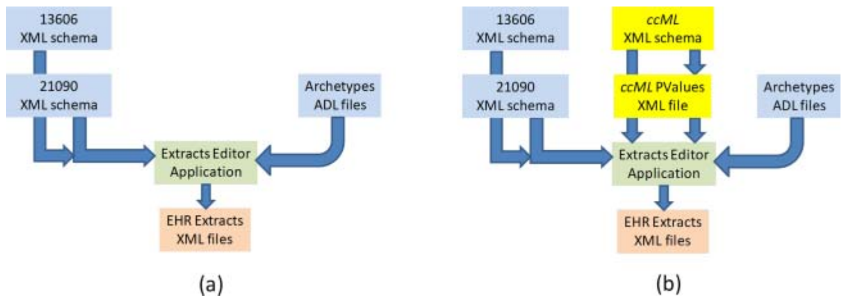


Figure 3 The extracts editor application with its input files and electronic health record (EHR) extracts output, without the ccML module (A) and after the inclusion of this module (B). This figure is only reproduced in colour in the online version.

ccML PVCodes

Much of the semantics of ccML is entailed by its *code* attribute. The *code* attribute can be defined for every element of the language (*PVType*, *PVComplexElement*, *PVSimpleElement* and *PVAttribute*) and it can adopt seven different values specified in enumeration type *PVCodes*. These are the code values that solve the necessities arising during the process of edition of an EHR extract, like those presented in the previous example. In this section a brief description of the motivation and the semantics of each value are provided.

- *pv_none*: this value indicates that the element that contains the attribute code has no special semantics and thus the interpreter has nothing to do with it. However, its descendants normally can have an attribute code with other significant values.
- *pv_value*: this code indicates that the *PVAttribute* containing it has a PV given by the value of the code in the ccML file and can be assigned automatically by the extracts editor to it.
- *pv_schema*: this value means that the (simple) element containing the attribute code has a fixed value. This value is stored in the 13606 XML schema in the fixed attribute of that element and not in the ccML file. The extracts editor must go to that schema, fetch the value and assign it automatically to the element on building the extract.
- *pv_now*: this code indicates that the (simple) element that contains it has a computable value, which is the current date and time in the format specified by class *DateTime*. This value must be computed by the editor and stored automatically as the value of the element.
- *pv_unique*: this value means that the *PVAttribute* containing it has a value that must be a unique string. The editor system is responsible for its computation guaranteeing its uniqueness inside the system. Uniqueness across systems is normally guaranteed through the use of the attribute *root* (see example in the previous section).
- *pv_given*: this code means that the *PVAttribute* containing it has a value that must be obtained by indexation of a list of strings that is passed to the system during interpretation. This list contains values of strings that are 'given' to the system independently of the RM used.
- *pv_archetype*: this value indicates that the value of the *PVAttribute* containing the code depends on the binding of its element to an archetype.

RESULTS

ccML language interpretation

Figure 3B shows the architecture of an extracts editor application using our proposed solution. To the RM's XML schemas and the ADL archetype files used as input in figure 3A, the ccML schema and the ccML file are added as further input. The ccML file does not modify nor suppress a single piece of information provided by the RM schemas. It only provides the application with new pragmatic or convenient information that helps the editor to do its job better.

Figure 4 ccML language expressed as a single-type tree grammar. This figure is only reproduced in colour in the online version.

N = { PVSchema, PVComplexElement, PVSimpleElement, PVAttribute, PVType, PVCodes, PVschema, PVnow, PVvalue, PVunique, PVgiven, PVarchetype, PVnone, Name, Code, Type, Site, Value }

T = { pv_schema, pv_now, pv_value, pv_unique, pv_given, pv_archetype, pv_none, string }

S = {PVSchema }

P = { PVSchema → (PVComplexElement*, PVSimpleElement*, PVAttribute*, PVType*) ,

PVComplexElement → (PVComplexElement*, PVSimpleElement*, PVAttribute*, Name, Code?, Type)

PVSimpleElement → (PVAttribute*, Name, Code?, Type?)

PVAttribute → (Name, Code?, Site?, Value?)

PVType → (PVComplexElement*, PVSimpleElement*, PVAttribute*, Name)

PVCodes → (PVschema, PVnow, PVvalue, PVunique, PVgiven, PVarchetype, PVnone)

PVschema → pv_schema (ε), PVnow → pv_now (ε), PVvalue → pv_value (ε), PVunique → pv_unique (ε), PVgiven → pv_given (ε), PVarchetype → pv_archetype (ε) PVnone → pv_none (ε)

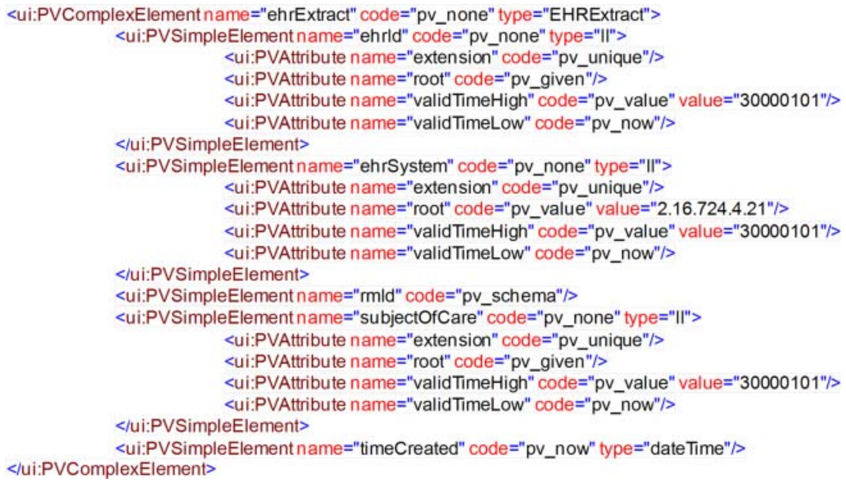
Name → string (ε)

Code → (PVCodes)

Type → string (ε)

Site → string (ε)

Value → string (ε) }



```
<ui:PVComplexElement name="ehrExtract" code="pv_none" type="EHRExtract">
        <ui:PVSimpleElement name="ehrId" code="pv_none" type="II">
                <ui:PVAttribute name="extension" code="pv_unique"/>
                <ui:PVAttribute name="root" code="pv_given"/>
                <ui:PVAttribute name="validTimeHigh" code="pv_value" value="30000101"/>
                <ui:PVAttribute name="validTimeLow" code="pv_now"/>
        </ui:PVSimpleElement>
        <ui:PVSimpleElement name="ehrSystem" code="pv_none" type="II">
                <ui:PVAttribute name="extension" code="pv_unique"/>
                <ui:PVAttribute name="root" code="pv_value" value="2.16.724.4.21"/>
                <ui:PVAttribute name="validTimeHigh" code="pv_value" value="30000101"/>
                <ui:PVAttribute name="validTimeLow" code="pv_now"/>
        </ui:PVSimpleElement>
        <ui:PVSimpleElement name="rmId" code="pv_schema"/>
        <ui:PVSimpleElement name="subjectOfCare" code="pv_none" type="II">
                <ui:PVAttribute name="extension" code="pv_unique"/>
                <ui:PVAttribute name="root" code="pv_given"/>
                <ui:PVAttribute name="validTimeHigh" code="pv_value" value="30000101"/>
                <ui:PVAttribute name="validTimeLow" code="pv_now"/>
        </ui:PVSimpleElement>
        <ui:PVSimpleElement name="timeCreated" code="pv_now" type="dateTime"/>
</ui:PVComplexElement>
```

Figure 5 A fragment of the file written in ccML used by the extracts editor application. This figure is only reproduced in colour in the online version.

As with any other language the semantics of ccML depends on the procedure used to interpret it.

During the edition task, when the user selects an element from the RM, for instance *ehrExtract::EHRExtract* (see figure 1), an extracts editor will need data like that presented in the example of the previous section, and similar problems to those described in that example may arise.

Here is where the usefulness of ccML appears. There are corresponding elements in the ccML file that can have PV, and the procedures of the extracts editor that compute the current node successors from the RM schema must pass through the filter of the PV's ccML file sampled in figure 5. It represents a *PVComplexElement* declaration with name 'ehrExtract' and type 'EHRExtract'. Please note that the ccML file mimics the structure of the RM, but it is only providing extra convenient information.

Sometimes it is necessary in ccML to state properties of entire data types, rather than just elements of some given type in the way explained above.

figure 5 and figure 6 show a ccML example of this possibility. In figure 6 several *PVCodes* for any element of data type *II* are defined in a *PVType* declaration (see its correspondent sample of the RM in figure 2) and in figure 5 several declarations of particular elements of this type are shown. The *PVCodes* asserted by the ccML fragment in figure 6 concern the uniqueness of the value of extension (ie, there may be many instances of elements with this attribute but all of them should have a different unique value) and the PV specified for the other three attributes. These declarations represent different restrictions on the elements of this specific data type. Notice once again that these descriptions do not affect the information provided by the RM fragment shown in figure 1.

DISCUSSION

Capturing ambiguity in the ccML single-type tree grammar

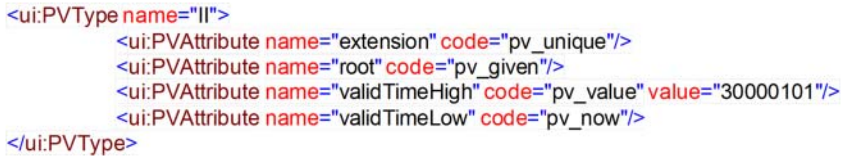
The previous example illustrates an important aspect of the interpretation of ccML. In particular, restrictions to simple element *ehrSystem::II* (of type *II*) are declared in ccML in figure 5 and several restrictions to generic type *II* are declared in figure 6. However, attribute *root* in the former has a different code, namely *pv_value* with a PV, because that *ehrSystem* comes from a computer in another system: ccML seems to be contradictory. In other words, if the application system encounters an *ehrSystem::II* element in the RM, which of the two ccML declarations should be obeyed?

The tree grammar developed in the previous subsection can be used to detect such ambiguities. Taking the same example, if the system encounters an element *ehrSystem::II* with four attributes declared in ccML (see figure 5), this element can match the production rule (figure 4) with the left-hand side *PVSimpleElement* with four *PVAttributes*, and a *Name* whose string value is 'ehrSystem' and a *Type* whose string value is 'II'. It can also match the production *PVType* with four *PVAttributes* and a *Name* whose string value is 'II'. Note that the converse is not true: when an arbitrary element (ie, with some given name and type 'II') is processed if it matches the *PVType* rule it does not in general match the *PVSimpleElement* production, because its arbitrary name does not always match the *Name* of this rule (which is obligatory), even when it matches the string *Type*.

Semantics in the ambiguity resolution

Having detected this kind of ambiguity in the ccML grammar the question is raised as to how to overcome it. The answer is quite straightforward: the interpreting module builds a system of priorities in which the *PVType* declarations have the lowest priority level. In other words, a *PVSimpleElement* (or a *PVComplexElement*) declaration prevails over a *PVType* declaration and there is no need for a different explicit declaration for every possible element/type combination.

The semantics intended by this interpretation is that a generic type declaration is left as a default case and should only function in the case that no specific element of that same type be declared with its own *PVCodes* elsewhere in the ccML file. Note that this is consistent with the criterion that more specificity should prevail over more generality. The opposite case would make the individual element declarations useless: the type declaration would prevail making every element of that type behave in the same way.



```
<ui:PVType name="II">
        <ui:PVAttribute name="extension" code="pv_unique"/>
        <ui:PVAttribute name="root" code="pv_given"/>
        <ui:PVAttribute name="validTimeHigh" code="pv_value" value="30000101"/>
        <ui:PVAttribute name="validTimeLow" code="pv_now"/>
</ui:PVType>
```

Figure 6 A declaration of restrictions on a generic type in ccML. This figure is only reproduced in colour in the online version.

Status report

The ccML language development during this research has been codified and tested as a substantial part of a complete Java standalone application implementing a visual ISO/EN 13606 EHR extracts editor. It serves as proof of concept of the new language introduced in this research paper.

Codification of the ccML language has implied both the writing of the XML schema and the *PV's* ccML files as separate input to the editor, and an additional Java layer on top of the editor application that filters and interprets the content of the latter. Both parts of the system have been developed and tested in several projects (see Acknowledgements).[24]

The Java filter layer fit very smoothly into the entire extracts editor application: the ccML file is preprocessed constructing its in-memory representation using its definition as given by its XML schema; at every moment that a new element is chosen during the editing process the filter layer additionally processes the ccML file in-memory representation and its context completion information triggers the automatic generation of new elements and their fulfillment. This process is transversal to the standard extracts edition process—it could be removed by simply providing an empty ccML file: the extracts edition would proceed exactly as if no context completion mechanism had been provided; or its behavior could be modified at run time adapting it to any particular situation or user preferences.

The RM's XML schemas used in this research were provided by Sun, Austin and Kalra, in ISO EN 13606 part 1 XML form version 3, University College London, 2011 (personal communication), and can be found in University College London.[11]

The entire application runs normally on standard workstation hardware over Windows, the expected ISO/EN 13606 elements and attributes triggered by the ccML file are generated satisfactorily and the response times of the whole system have been considered plainly acceptable.

CONCLUSION

A new XML-based mark-up language ccML has been designed in order to make the task of editing ISO/EN 13606 EHR extracts more practical. Files written in this new language are complementary to the 13606-21090 XML schemas and to the ADL files representing archetypes, and provide the necessary pragmatics information. This additional input to the application system greatly helps health professional users to edit an EHR extract. It automatically fills in many fields with values that otherwise should be entered manually by the user. This results in a process that is faster and safer.

The ccML language has been analyzed from the standpoint of formal language theory in order to understand better its characteristics and properties as a single-type tree language. In particular, this analysis serves to detect potential ambiguities in the grammar generating the language, and in the design of the ccML files in order to overcome those ambiguities. The ccML language is under development and new features can be added to it as new needs or possibilities of improvement arise, for instance visualization issues. The description given in this paper corresponds to the capabilities developed in research so far. New features added to ccML should be reflected in the grammar and checked against it in order to ensure the consistency of the language and its interpretation.

The ccML files have been created manually as the language design needs have emerged. However, it is also possible to develop in the future an authoring tool that assists in the creation of those files given both the ccML and the RM XML schemas. Such an authoring tool would help the system designer very considerably in developing and validating the ccML file corresponding to his or her needs given the RM that defines the specific information system application.

Another important characteristic of the ccML language is its independence from a specific RM. In this sense, and remaining in the EHR extracts edition context, this work could also be applied to HL7 RIM, the OpenEHR RM or to the results of the CIMI initiative once a model is agreed upon. But the ccML approach can in fact be applied to any application generating instances of models following a paradigm established by a RM represented by an XML schema and conforming to the dual model separation between information and knowledge, ie, using the archetypes framework. These applications are not restricted to EHR extracts edition and not even to health informatics but may be extended to any other task involving the generation of instances of models designed using a RM. This is a field with broad applicability.

Acknowledgments The authors would like to thank Dr Dipak Kalra, leader of the EHRCom task force that defined the ISO/EN 13606 standard and his team from University College London for their kind permission to use the ISO/EN 13606 W3C XML schema.

Funding This research has been partially supported by projects PI09/90110 'Plataforma de innovación en nuevos servicios de telemedicina y eSalud para pacientes crónicos y dependientes (Innovation platform for new telemedicine and e-health services for chronic and dependent patients)—PITES' and PI08/1148 'Representación de información clínica en cáncer de mama mediante arquetipos (Modeling clinical information in breast cancer through archetypes)—CAMAMA' from Fondo de Investigación Sanitaria (FIS) Plan Nacional de I+D+i, and by project CEN-20091043 'Tecnologías disruptivas para la rehabilitación del futuro (Disruptive technologies for Rehabilitation in the Future)—REHABILITA' from Plan Nacional de Investigación Científica Desarrollo e Innovación Tecnológica—CENIT-E.

Competing interests None.

Provenance and peer review Not commissioned; externally peer reviewed.

Open Access This is an Open Access article distributed in accordance with the Creative Commons Attribution Non Commercial (CC BY-NC 3.0) license, which permits others to distribute, remix, adapt, build upon this work non-commercially, and license their derivative works on different terms, provided the original work is properly cited and the use is non-commercial. See: http://creativecommons.org/licenses/by-nc/3.0/

REFERENCES

1. **Commission of the European Communities—COM.** 356: e-Health—making health care better for European citizens: An action plan for a European e-Health Area. European Union, Publications Office Brussels, 2004-04-30 2004.
2. **US Department of Health and Human Services.** Development and Adoption of a National Health Information Network (NHIN) Request for Information, Nov. 09, 2004, p.2. http://www.hhs.gov/healthit/rfi.html (accessed Aug 2012).
3. **Bakken S,** Campbell KE, Cimino JJ, *et al*. Toward vocabulary domain specifications for health level 7-coded data elements. *J Am Med Inform Assoc* 2000;**7**: 333–42.
4. **Beale T.** Archetypes: constraints-based domain models for future-proof information systems 2002. http://www.openehr.org/publications/archetypes/archetypes_beale_oopsla_2002.pdf (accessed Aug 2012).
5. **ISO.** International Organization for Standardization. http://www.iso.org/iso/home.htm (accessed Aug 2012).
6. **CEN.** European Committee for Standardization http://www.cen.eu/cen/Pages/default.aspx (accessed Aug 2012).
7. **Health Level Seven.** Health Level Seven International. http://www.hl7.org (accessed Aug 2012).
8. **OpenEHR.** openEHR. http://www.openehr.org (accessed Aug 2012).
9. **CIMI Wiki.** Clinical Information Modeling Initiative. http://informatics.mayo.edu/CIMI/index.php/Main_Page (accessed Aug 2012).
10. **Kalra D,** Lloyd D. *ISO 13606 electronic health record communication part 1: reference model*. ISO 13606-1. Geneva: ISO, 2008.
11. **University College London.** UCL Centre for Health Informatics & Multiprofessional Education (CHIME). ISO EN 13606 Schema. http://www.ehr.chime.ucl.ac.uk/code/schema_2.0/ISOEN13606_2.0.zip (accessed Aug 2012).
12. **Kalra D,** Beale T, Lloyd D, *et al*., eds. *Electronic health record communication part 2: archetype interchange specification*. ISO 13606-2. Geneva: ISO, 2008.

13. **EN13606 Archetype Definition Language files**. http://sres.saude.mg.gov.br/arquetipo/listar (accessed Aug 2012).
14. ISO 21090:2011 Health informatics Harmonized data types for information interchange. International Organization for Standardization. Geneva, Switzerland. 2011.
15. **Muñoz A,** Somolinos R, Pascual M, *et al*. Proof-of-concept design and development of an EN13606-based electronic health care record service. *J Am Med Inform Assoc* 2007;**14**:118–29.
16. **Austin T,** Su Lim Y, NGuyen D, *et al*. Design of an electronic healthcare record server based on part 1 of ISO EN 13606. *J Healthc Eng* 2011;**2**:143–60.
17. **Base de Registro Eletronico em Saúde do estado de Minas Gerais (Brasil)**. Base de Registro Eletrônico em Saúde. http://sres.saude.mggov.br (accessed Aug 2012).
18. **CeHis project.** Center för eHälsa i samverkan http://www.cehis.se/en (accessed Aug 2012).
19. **Beale T,** Heard S. The openEHR archetype model archetype definition language ADL 1.5 2010. Available at: http://www.openehr.org/wiki/pages/viewpage.action?pageId=196633 (accessed Aug 2012).
20. **Murata M,** Lee D, Mani M, *et al*. Taxonomy of XML schema languages using formal language theory. *ACM Trans Internet Technol* 2005;**5**:660–704.
21. **Thompson HS,** Beech D, Maloney M, *et al*., eds. *XML Schema Part 1: Structures 2000*. http://www.w3.org/TR/xmlschema-1/. (accessed Aug 2012).
22. **Common H,** Dauchet M, Gilleron R, *et al*. Tree Automata Techniques and Applications 1997. http://tata.gforge.inria.fr/. (accessed Aug 2012).
23. **Takahashi M.** Generalizations of regular sets and their application to a study of Context-Free Languages. *Inf Control* 1975;**27**:1–36.
24. **REHABILITA research project 2010**. Rehabilita. http://rehabilita.gmv.com/web/guest. (accessed Aug 2012).