HIC 2006 Bridging the Digital Divide: Clinician, consumer and computer
Eds: Johanna Westbrook & Joanne Callen
Publisher: Health Informatics Society of Australia Ltd (HISA)
ISBN 0 9751013 7 4

A Practical Implementation of a Two Level Archetype Based Clinical Model

David Ashton

Meridian Health Informatics, Surry Hills NSW 2010

ABSTRACT

The concept of archetypes (Thomas Beale [1]) and their application to clinical informatics is now recognised as a powerful model for designing EMR and EHR systems. There have been a number of practical developments based on this concept, most recently by DSTC in Queensland where an EHR trial is being conducted for chronic disease management. Earlier attempts have also been published (Ashton [2]) as well as other projects currently in progress. This paper will describe a new two level schema for a clinical database engine based on archetypes. It will describe how this engine will allow subject matter experts to implement EHR and EMR systems, directly, without programmers. The elements of the engine include an interface to define content (i.e. archetypes, structures and rules), referred to as a 'modeller' and an end-user interface to enter and access clinical information is referred to as a 'validator/browser'. The paper will describe how the unique design of these components allows expert clinicians to take ownership of their clinical applications. It will also describe the application of this technology to the development of a new NSW, state wide clinical obstetric system (ObstetriX) that commenced live operation in August 2004.

Background

A clinical obstetric system (Obstet) based on two level model architecture was developed by the author for Westmead hospital in 1992. The design was the subject of a paper published in 1996 [2] and reviewed by Associate Professor C. J. (Kit) Dampney in 1998 [3]. The Obstet system was developed using the Cognos development tool PowerHouse for operation in the Digital Vax/Alpha mainframe environments. In 1995 NSW Department of Health (DoH) formed a consortium of clinical users and Obstet was implemented in the major public teaching hospitals in NSW.

With the phasing out of mainframe technology, in 2003 the NSW DoH granted funding for the development of ObstetriX to replace Obstet. ObstetriX was developed using Microsoft .NET technology and provided a wider range of deployment, interface and integration options. The ObstetriX project was completed in July 2004 and is in the process of implementation throughout NSW.

Obstet and ObstetriX are both comprehensive Electronic Medical Record (EMR) systems that store longitudinal clinical records for birthing women from first presentation through to delivery and mother and baby discharge. These systems are an important element in patient management and a valuable epidemiological resource for research purposes.

HIC 2006 Bridging the Digital Divide: Clinician, consumer and computer
Eds: Johanna Westbrook & Joanne Callen
Publisher: Health Informatics Society of Australia Ltd (HISA)
ISBN 0 9751013 7 4

Today around 50% of all births in NSW are recorded on Obstet and ObstetriX with records on over 300,000 women. The system is used by thousands of clinicians in 30 birthing centres throughout the state. This paper discusses the design principals used to develop both the Obstet and ObstetriX applications.

Two level models

The idea behind using a two level model to describe a system schema is not new. There have been a variety of systems developed, which make use of meta-meta data models. However Thomas Beale (Beale et al [1]) introduced a new terminology and identified the design elements needed to build robust applications, which are capable of meeting a wide range of user needs.

To understand the benefits of using a two level model it is useful to firstly review the conventional single level model design approach. This approach to system design is usually introduced very early in most undergraduate software engineering courses.

Classical approach to database system design

The single level data model is derived through the familiar process of data analysis called normalisation[1]. Data items that relate to the subject are collected and grouped together to form relations. These relations, and their attributes, are most frequently described in a relational database schema and are called 'meta-data'. The principle of normalisation promotes the recognition of cardinality[2] between entities and dependencies and has the effect of reducing redundancy but at the same time increasing the number of relational tables in the database (C.J. Date [6]). There are various levels of normalisation attainable but in all cases the process results in more relational tables specified within the schema.

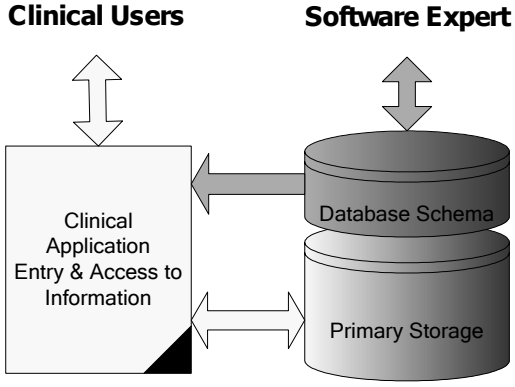


Fig 1 Classical Design

What is the effect of normalisation on program component design? The aim is to reduce data redundancy and make program modules more robust by increasing code/data independence. Experience suggests however that when application design is focussed on tables then the number of maintainable program elements as well as their complexity *is an increasing function of the number of tables* rather than just the number of data attributes. Furthermore the complexity introduced by

[1] Normalisation is a set of techniques for organising data into tables so as to eliminate certain types of redundancies. Each attribute in a normalised entity is functionally dependent on the value of the whole key and no other attribute in the model.

[2] Cardinality is a term indicating the number of objects that can be related to another object. For example a mother (object) may have zero or more children (objects). The 'zero or more' defines the cardinality between mother and child.

HIC 2006 Bridging the Digital Divide: Clinician, consumer and computer
Eds: Johanna Westbrook & Joanne Callen
Publisher: Health Informatics Society of Australia Ltd (HISA)
ISBN 0 9751013 7 4

additional tables is not offset by the reduction in data redundancy. This assertion is supported by traditional 'function point' analysis (A.J.Albrecht [5]).

The process of data model normalisation therefore increases application complexity and hence, development and maintenance time and cost. Most importantly it means that changes to the data model, either through new attributes or changed cardinality will involve changes to application programs and professional developers in most instances would need to carry out these changes.

Data models in health related fields tend to contain significant numbers of attributes with often unstable or volatile cardinality. Furthermore many of the data attributes, to be useful for aggregate analysis, must be 'classified' or coded data. This means that these attributes need reference tables, maintenance functions and version control. It follows that clinical systems, which are based on conventional single level models, have large complex data models that are expensive to build and maintain.

Two Level Model Schema Structure using Archetypes

The concept behind a two level data model is the introduction of a level of abstraction between the program logic and the database schema (See figure2). This second model level focuses on individual, self contained, clinical attributes that are independent meta descriptions of clinical information such as 'blood pressure', 'mode of delivery' and 'birth weight'. This model expresses the character of these clinical data attributes and stores this information as data in the database rather than in the database schema. Additional software is needed to manage this 'meta' data and we name this application the 'modeller'.

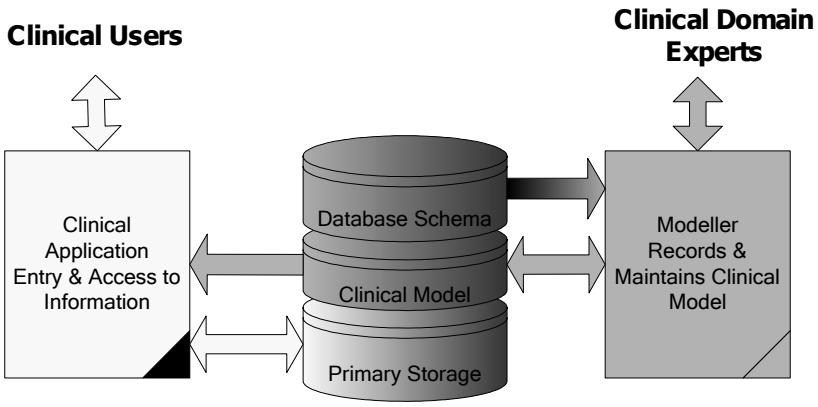


Fig 2 Two Level Clinical Archetype Design

Central to the 'two level' model is the clinical attribute class or 'archetype'. This class is a 'meta' definition of a clinical fact. Examples of instances of this class would be:

Table 1 Archetype examples

Archetype Name	Mnemonic
'Mode of Delivery'	ModeofDelivery
'Apgar Score 5 min'	ApgarScoreatFiveMinutes
'Blood Group'	BloodGroup
'Birth Weight'	BirthWeight
'Stage One Onset'	StageOneOnset

HIC 2006 Bridging the Digital Divide: Clinician, consumer and computer
Eds: Johanna Westbrook & Joanne Callen
Publisher: Health Informatics Society of Australia Ltd (HISA)
ISBN 0 9751013 7 4

'Cervix Dilation'	CervixDilation

The mnemonic column is a convenient handle by which we can reference an archetype. An Archetype is always related to a Domain, which are the set of constraints we apply when assigning actual clinical values to the archetype. The descriptors that make up the Archetype and Domain are referred to as a 'class' definition. Some properties of the Archetype class and the Domain sub class include:

Archetype Class

- → Mnemonic
- → Name
- → Cardinality
- → Description
- → User Help
- → Narrative
- → Life span
- → Authority
- → Version
    - → Domain class
    - → Mnemonic
    - → Description
    - → Value Type
    - → Event Value
    - → Value
    - → Help
    - → Narrative
    - → Thread class

Because archetypes are concepts or meta data, instances of these classes are actual user values that can be associated with a patient:

Table 2 Archetype instance examples

Archetype Mnemonic	Patient Instance #1	Patient Instance #2
ModeofDelivery.Description	'Forceps'	'Vaginal Delivery'
ApgarScoreatFiveMinutes.Value	<null>	7
ApgarScoreatFiveMinutes.Description	'Not recorded'	'Score'
ApgarScoreatFiveMinutes.Type	integer	integer
BloodGroup.Description	O+	<null>
BirthWeight.Value	3455	<null>
BirthWeight.Description	'Grams'	'Not recorded'
StageOneOnset.EventValue	12-5-2005 12:10 am	12-5-2005 12:10 am
CervixDilation.Value	5	<null>
CervixDilation.Description	'Cm'	'Not recorded'

HIC 2006 Bridging the Digital Divide: Clinician, consumer and computer
Eds: Johanna Westbrook & Joanne Callen
Publisher: Health Informatics Society of Australia Ltd (HISA)
ISBN 0 9751013 7 4

ScalpPH.Value	3.4	2.8

As illustrated in these examples, when we create an instance of an archetype class it needs to be represented by an object which could have many properties. In the ObstetriX implementation we treat an archetype instance as a 'composite object' that can be referenced programmatically by a unique mnemonic.

For the purpose of creating instances of archetypes and attaching those instances to actual patient instances we need to introduce two organisational classes. These are 'Groups' and 'Threads' and they allow us to form practical collections of archetypes that can be referenced in a program script. Collections of archetypes are organised in the Thread class and collections of Threads organised in the Group class.

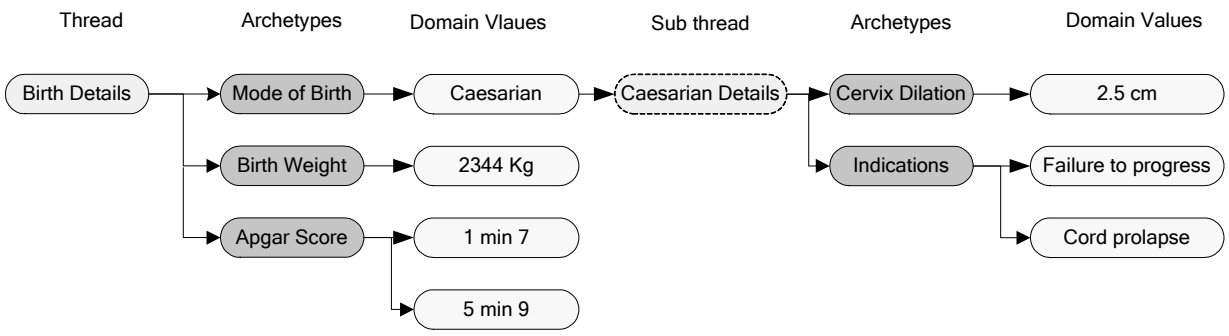
Some properties of Thread and Group classes are:

Group Class

- → Description
- → Help
- → Date
- → Group Class
- → Thread class
  - → Description
  - → Help
  - → Cardinality
  - → Authority
  - → Archetype class

Groups can link to collections of other group classes (eg a patient could have many pregnancies and many babies). And archetypes can also link to threads via the Domain value class. For example, an instance of the archetype 'mode of delivery' could have a 'value' 'Caesarean Section'. This would then be associated with a further thread class consisting of a collection of archetypes describing the caesarean procedure and the reasons for it. Fig 3 illustrates this association between archetype instances and threads and domain values.

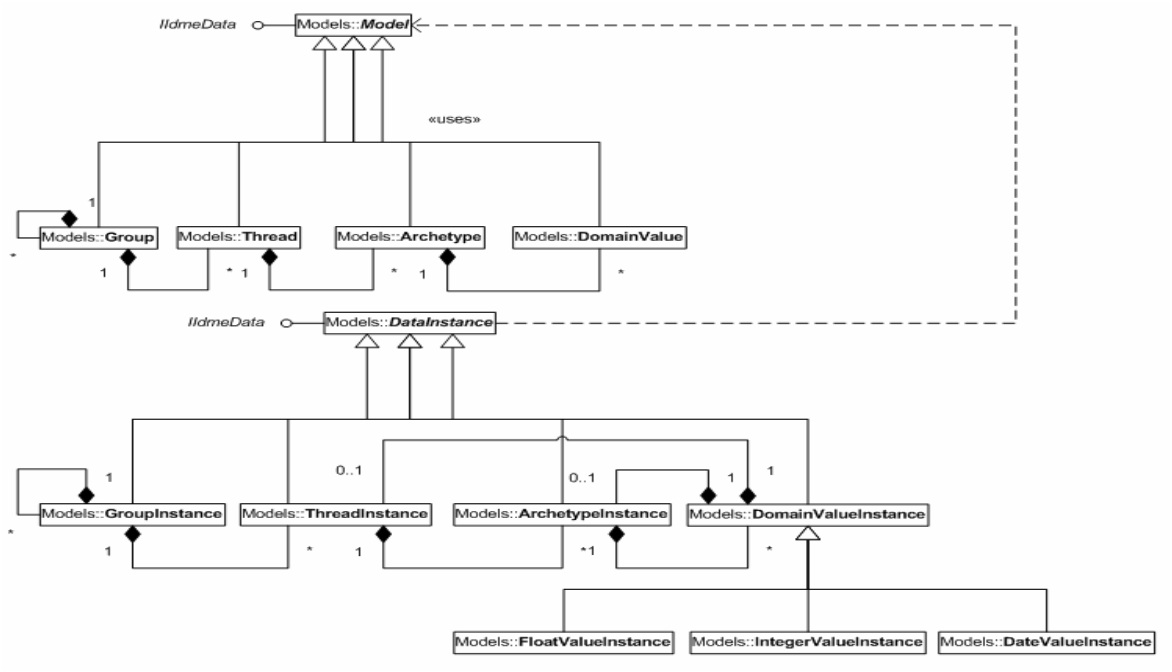
Fig 3 Thread and sub-thread relationship



The data model for storing archetype and archetype instances is shown in Fig 4

HIC 2006 Bridging the Digital Divide: Clinician, consumer and computer
Eds: Johanna Westbrook & Joanne Callen
Publisher: Health Informatics Society of Australia Ltd (HISA)
ISBN 0 9751013 7 4

Fig 4 Archetype data model structure



This model consists of only eight primary entities. The Group, Thread, Archetype and Domain Value comprise the clinical meta model. We refer to this as the 'Archetype Model'. The corresponding instance entities Group instance, Thread instance, Archetype instance and Domain value instance represent the meta descriptions of the actual data. This is referred to as the 'Reference Model'. *It is significant that the meta-meta schema is completely generic and contains no clinical subject matter.*

Normalisation & Cardinality with Archetypes

Unlike many financial and administrative software applications, clinical systems often have thousands rather than hundreds of data attributes with often complex and volatile cardinality and relationships. One of the most significant advantages of building applications using archetypes is that no normalisation of the clinical data model need take place at all. This is because *the archetype instance is a maximally normalised object* in so far as there are no tables and the archetype stands alone.

The thread and group structures allows a designer to organise archetypes into a convenient assembly to permit a logical navigation and process flow for users. This means that assigning values to archetype instances in a particular way will lead to other archetypes or groups of archetypes. There is no 'relational' association between archetypes or tables structures that you might find in an SQL database. If a situation exists where an archetype can have many instances this constraint can be set as a property of the Archetype.

For example the archetype 'Analgesia in stage one of labour' can have more that one instance for a woman giving birth, hence for this archetype, the cardinality property can be set to allow many instances to be created. The Archetypes in the table below illustrate this property.

In a similar way, cardinality can be expressed as a property of a Thread. This allows collections of archetype instances, attached to the thread, to be created. An example would be a thread of archetype instances created each time a patient attended a clinic. In this case the Thread would need to have cardinality greater than one with respect to the patient entity.

HIC 2006 Bridging the Digital Divide: Clinician, consumer and computer
Eds: Johanna Westbrook & Joanne Callen
Publisher: Health Informatics Society of Australia Ltd (HISA)
ISBN 0 9751013 7 4

Table 2 Archetype instance examples

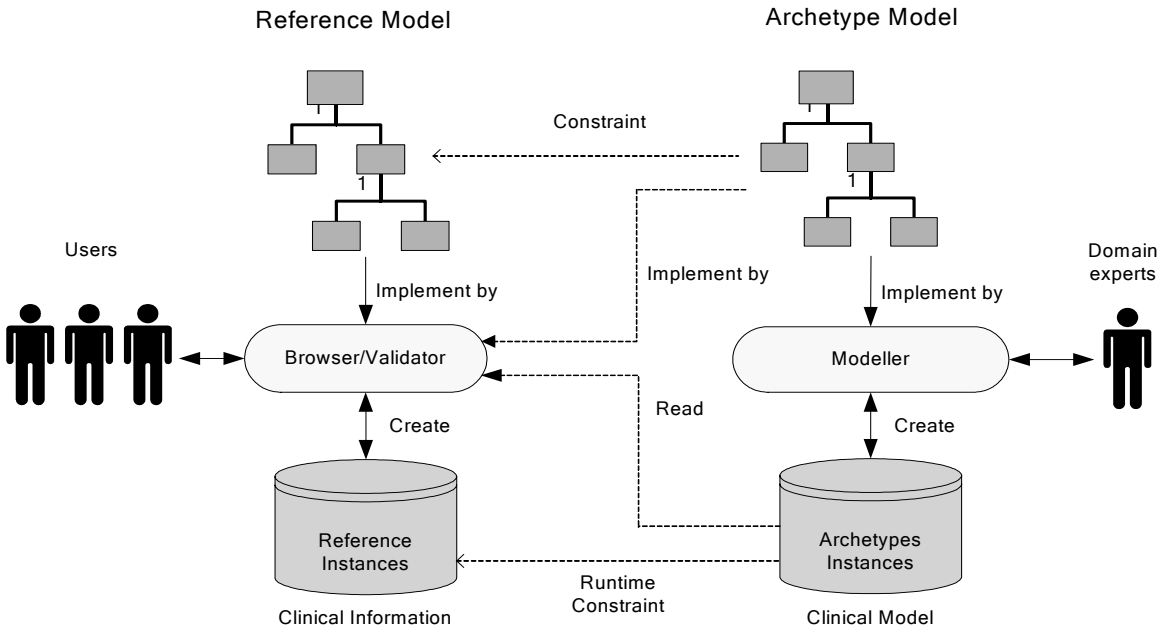
Archetype	Archetype Instance #	Patient Instance #1	Patient Instance #2
'Analgesia in stage one of labour'	1	Nitrous oxide gas	General
	2	Psychoprophylaxis	
	3	Epidural	
'Obstetric indication/s CS'	1		Poor progress in labour >3cm
	2		Cord prolapse

Application Component Design

The use of the 'two level' approach to system building profoundly changes the way an application is designed. Two application modules are required. The 'modeller' application is used by domain experts to create and maintain the instances of the archetype model. A 'domain expert' is most likely to be a user who understands the specific information environment and business processes. End-users use the 'validator/browser' to create and maintain instances of the Reference Model that represent the actual data (see Fig 4).

As mentioned above, there is no longer any subject matter stored in the database meta schema. This information is now stored as data (instances of the archetype model). It follows that there is no longer any subject matter stored in lines of code of either the user application (Browser/Validator) or the Modeller applications. This makes the application software completely independent of data.

Fig 5 The Archetype Process Model



Unlike most conventional computer applications *the size and complexity of either application is no longer dependent the number of subject related data attributes*. Fig 5 illustrates the relationship of process components to storage and meta-data components. This therefore is a method of building

HIC 2006 Bridging the Digital Divide: Clinician, consumer and computer
Eds: Johanna Westbrook & Joanne Callen
Publisher: Health Informatics Society of Australia Ltd (HISA)
ISBN 0 9751013 7 4

large systems with highly generic components that can be extended without changing the central application elements.

User Interface for an Archetype based system

In the previous section we described how our application creates instances of data from the Reference model using the constraints applied from the Archetype model. All subject matter is stored as data so our user application, the Browser/Validator, will be highly generic and can be independent of clinical definitions. It will maintain instances of group, thread, archetype and domain values and will be linked to a patient instance. The main functional components required to support the interface are:

Patient instance maintenance; the archetype model needs to be linked to an actual patient entity. In the case of ObstetriX this is a patient, a pregnancy or a baby entities. This module manages the linkage of the reference model components these entities.

User path navigation; archetypes are organised by groups and threads according to clinical and organisational needs. This component is implemented as a tree view and allows users to navigate through the groups, threads and archetypes. The structure between the patient entity and the archetype is referred to as the 'path'. The navigator tree view is driven primarily by archetype instance data.

Archetype instance maintenance; collections of archetypes are conveniently organised into threads, and these form the atomic maintenance unit in the ObstetriX system. This component allows users to create an instance of the Reference model according to the constraints in the Archetype model; that is, the actual recording of clinical information.

The organisation of the ObstetriX clinician interface is show in fig 5. The left panel is a tree view that organises archetypes using Groups and Threads. The central panel displays the list of archetypes attached to the thread that is in focus on the tree view, if these have actual data instances associated with them, then that data is also displayed. Multiple instances of archetypes appear as multiple rows on this list. Archetype properties including descriptions and help messages provide assistance to users. The right hand column contains the domain values possible for the archetype that is in focus in the central panel. If the archetype has a value already then a 'tick' will appear next to that value. The domain value panel also contains links to sub-threads and possible extension values including text, numeric values and dates. If the domain value is classified as an event, then it will always be associated with a date and time.

Recursive group structures are provided to organise threads. These structures are also used in the tree view to link groups (or threads) to patient entities. The patient instance entities, patient, pregnancy and baby, appear as 'pseudo groups' in the tree view structure. When a patient entity group is in focus, then the attributes of that entity are made visible on the right hand panel of the window.

HIC 2006 Bridging the Digital Divide: Clinician, consumer and computer
Eds: Johanna Westbrook & Joanne Callen
Publisher: Health Informatics Society of Australia Ltd (HISA)
ISBN 0 9751013 7 4 Path

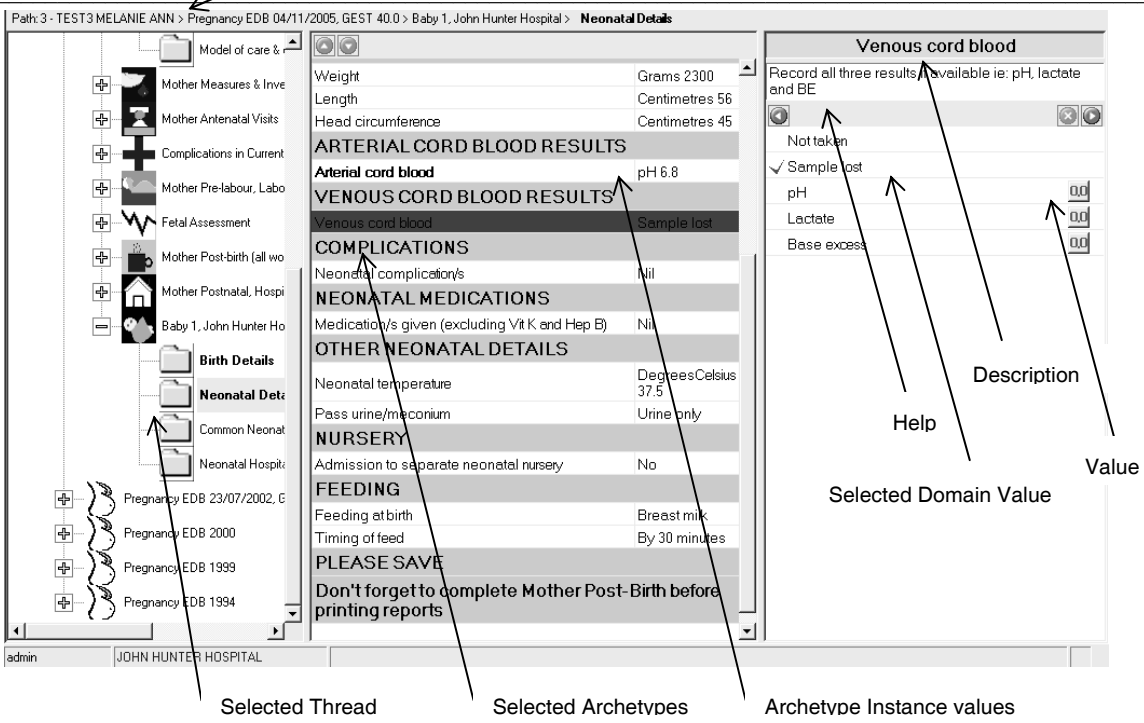


Fig 6 ObstetriX User Interface

The same generic format is used for every Thread (see Fig 6). And the user interface for each archetype is the same in all cases irrespective of whether the instance is a classification, text or quantitative information. The selection of a domain item may also initiate a branch to a sub-thread. While Figure 5 presents a GUI interface, an equivalent web interface is also available in ObstetriX.

While conventional systems may use a mixture of text fields, radio buttons, drop down lists and select boxes, this style of interface has been found to be most attractive to users as it is consistent, easy to learn and easy to obtain help when required. Users can focus on content rather than how to 'operate' the interface.

As each thread is completed the user is prompted to enter a personal identification code before the thread can be update. Once updated, the thread name in the tree view will be set to bold. If at any stage an archetype instance within a thread is changed, then the user ID is also recorded and a log is retained of all changes made at the archetype level.

Archetype Modeller Interface and Information Context

The archetype modeller is designed to allow the expert users to specify the content of the system in terms of the building blocks referred to in the previous sections. These are the groups, threads archetypes and domain values. The modeller interface allows the designer (domain experts) to build the tree view navigation structure that will enable the user to locate the threads and archetypes for a patient entity. Archetypes and their properties and domain values can then be specified and linked to threads.

The layout of the modeller interface is shown in Fig 7. The left hand panel allows the user 'navigation tree view' to be constructed, while the right hand windows provide for the specification of archetype properties.

Designers can apply a security profile for either an archetype or a thread depending on the level of privacy needed for the data. A security constraint applied to the thread will apply to all the

HIC 2006 Bridging the Digital Divide: Clinician, consumer and computer
Eds: Johanna Westbrook & Joanne Callen
Publisher: Health Informatics Society of Australia Ltd (HISA)
ISBN 0 9751013 7 4

archetypes in the thread. Security constraints are role based with users able to access information on a 'need to know' basis. Roles can be specified to be time dependent, based on the period of care episode.

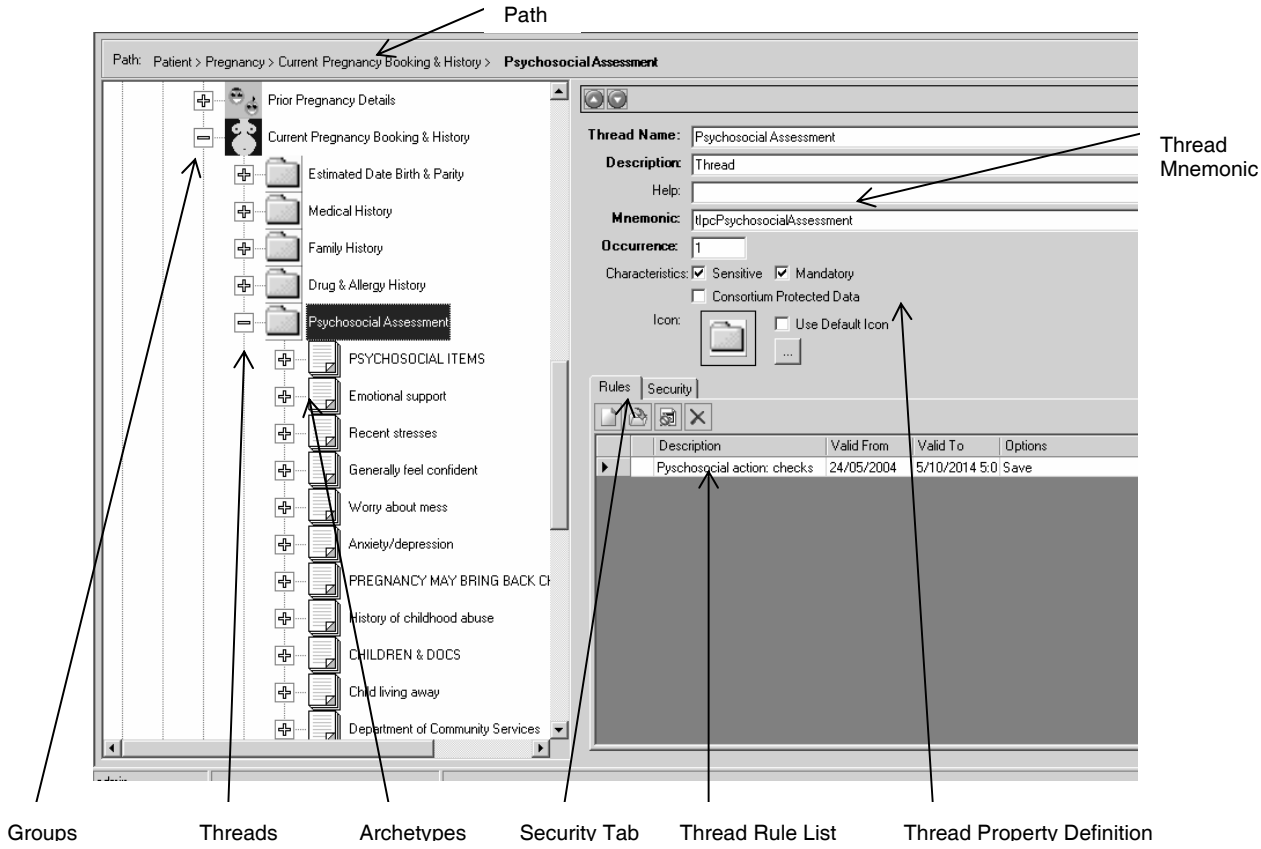


Fig 7 Modeller Interface

A central idea behind the concept of archetypes lies in the meaning of the term 'archetype'. *A fragment of data is only meaningful if we understand the context in which it was recorded.* The words literal meaning is 'original design'. If we create an instances of an archetype we are creating information based on a particular model or 'class', and that information can only ever have meaning in terms of that class definition. If we change that class, the instances or data created according to the original class can only have meaning when associated with the original class or 'archetype'.

This is a profoundly important concept so far as clinical data is concerned. When we create data based on an archetype, for this data to be meaningful we must retain an exact copy of that archetype at the time the data was recorded. If we wish to change the archetype, we must make a copy and create a new version. The archetype version is a property of the archetype and domain value class. Version maintenance is a critical function of the modeller interface.

The interface component used to create a new version of an archetype is show in Fig 8. Versions are controlled through non-overlapping date time ranges which allow new version to be activated at a planned point in time. All archetype instances created before the activation date and time will be created according the version currently active, after the activation date, the new version is used. Information is viewed in the context of the archetype version in existence when the data was recorded.

HIC 2006 Bridging the Digital Divide: Clinician, consumer and computer
Eds: Johanna Westbrook & Joanne Callen
Publisher: Health Informatics Society of Australia Ltd (HISA)
ISBN 0 9751013 7 4

The edit window shown in fig 8 will create a new version of the archetype if existing data instances of that archetype are found. The example 'Mode of Delivery' is a complex archetype with a scope that includes a number of sub threads.

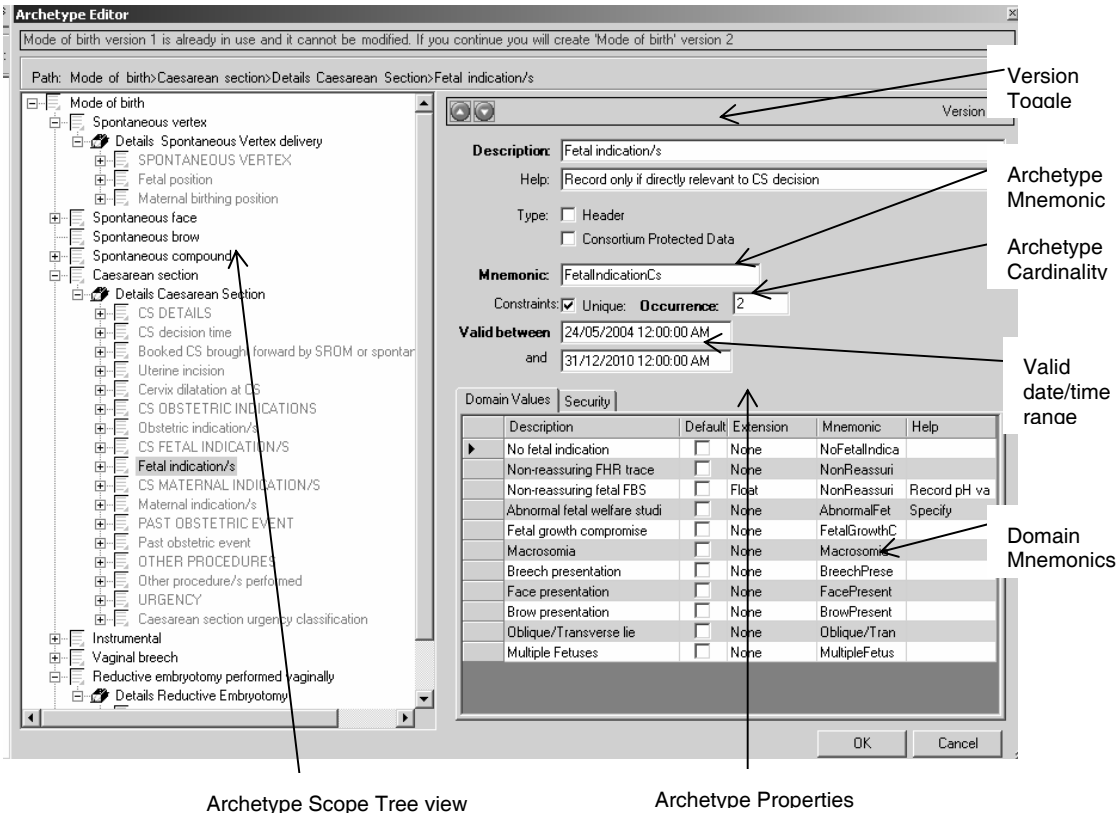


Fig 8 Archetype Editor 'Mode of Birth'

Archetype Relationships and Rules

The incorporation of archetype-based systems into real world computer applications leads us to consider methods of referencing archetypes from programming languages. While the creation of data instances conforms to the constraints of the Archetype model, there may also be clinical rules that need to be evaluated using one or more other data instances that may also be in existence. For clinical applications these data may have been entered at different times during the continuum of care for a patient. Compliance rules may need to be applied to any property of an archetype instance including classifications as well as value extensions.

In clinical applications, the requirement for procedural logic originates from the need to:

- Validate user entered data and prevent users from entering conflicting or invalid data,
- Assess clinical data and inform, warn or alert users of adverse clinical factors,
- Provide convenient summaries of clinical information,
- Perform calculations on quantitative data or calculate a value for display,
- Change archetype navigation depending on the value of entered data instances,
- Set and reset archetype instance values depending on entered data.

HIC 2006 Bridging the Digital Divide: Clinician, consumer and computer
Eds: Johanna Westbrook & Joanne Callen
Publisher: Health Informatics Society of Australia Ltd (HISA)
ISBN 0 9751013 7 4

In the ObstetriX example a rules language compiler has been integrated into the Modeller. Rule procedures are encapsulated modules that can be activated by group, thread or archetype manipulation (see Fig 9).

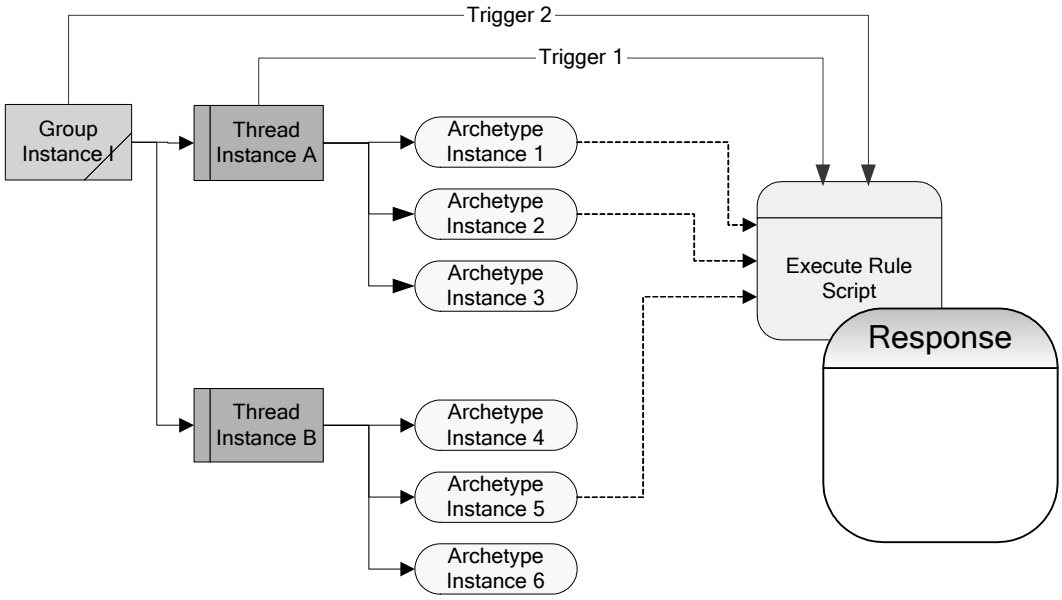


Fig 9 Integration of Rules Scripting in ObstetriX

Modeller Rule code is a full function declarative language with structural similarities to Microsoft C#. The Rule parser generates a compact execution module that can be triggered at runtime from specific user actions on groups, threads and archetypes.

Archetypes are referenced programmatically by the thread, archetype or domain mnemonics using the familiar object notation 'dot' syntax. The general form is:

For Archetype object properties

[<*thread mnemonic*>.<*archetype mnemonic*>.<*version*>]<*property*>

For Domain Value object properties

[<*thread mnemonic*>.<*archetype mnemonic*>.<*version*>.<*domain mnemonic*>]<*property*>

The inclusion of the thread mnemonic is optional and is added to improve the readability of the code.

An example of the initial version of an archetype object reference for the mode of birth archetype is:

[BabyBirthDetails.ModeBirth.1]

This table lists some of the properties of this archetype object and their values:

Property reference	Type	Value
1. [ModeBirth.1]Help	string	'Indicate the baby mode of birth'
2. [ModeBirth.1]Name	string	'Mode of Birth'
3. [ModeBirth.1]Exists	boolean	TRUE if an instance exists this archetype
4. [ModeBirth.1]Cardinality	Integer	1

HIC 2006 Bridging the Digital Divide: Clinician, consumer and computer
Eds: Johanna Westbrook & Joanne Callen
Publisher: Health Informatics Society of Australia Ltd (HISA)
ISBN 0 9751013 7 4

5. [ModeBirth.1]SelectedItemName	string	Contains the domain value name of the instance of the archetype
6. [ModeBirth.1]SelectedItemValue	Any	Contains the domain extension of the instance of the archetype entered by the user
7. [ModeBirth.1]Count	Integer	Number of occurrences of the archetype instance
8. [ModeBirth.1]Visible	boolean	TRUE, if the archetype is made visible, within the thread, to the user. Can be set by rules logic.

Note that property 1,2 and 4 are static properties that return values whether an archetype object exists or not. 3, 5, 6, 7 and 8 return values that depend on the existence of a particular archetype instance for a patient.

This table list some of the properties of the domain value object [NeonatalDetails.BirthWeight.1.Grams]

Property reference	Type	Value
1. [BirthWeight.1.Grams]Exists	boolean	TRUE if a birth weight was entered
2. [BirthWeight.1.Grams]Value	integer	The weight of the baby in grams
3. [BirthWeight.1.Grams]Name	string	'Grams'
4. [BirthWeight.1.Grams]Help	string	'Specify the baby weight in grams. Valid range is 800 to 5000 grams'
5. [BirthWeight.1.Grams]Text	string	Combination of Name and Value property. Eg '2300 Grams'
6. [BirthWeight.1.Grams]Count	int	Number of occurrences of the domain value instance
7. [BirthWeight.1.Grams]Visible	boolean	TRUE, if the domain value is made visible to the user. Can be set by rules logic.

As with the archetype properties, 9 and 10 are static and the remaining properties are dependent on the existence actual user data.

The following rule fragments illustrate how logic can be used to express rules between archetypes:

The rule prints an informative message to the user that the mother, having given birth to her first child may be suitable for Anti-D medication if the baby's rhesus factor is negative. Note the use of the 'Local( )' function that restricts the scope of the archetype object evaluation to the thread from where the rule is triggered.

```
if ([AnyPriorPregnancy.1].Exists and
   [AnyPriorPregnancy.1.No].Exists and
   Local([RhesusFactor.1.Negative]).Exists)
```

HIC 2006 Bridging the Digital Divide: Clinician, consumer and computer
Eds: Johanna Westbrook & Joanne Callen
Publisher: Health Informatics Society of Australia Ltd (HISA)
ISBN 0 9751013 7 4

Write("Is this Rhesus negative primipara suitable for Anti-D prophylaxis?");

We have discussed that a key advantage of archetype-based systems is that we can accommodate complex cardinality because the archetype object is maximally normalised. Depending on the cardinality of the archetype or thread and the scope of our rule, we must consider certain archetype instance properties to be of type array. Using the rule parser we can express an indexed archetype or domain value object in the usual way:

[<*thread mnemonic*>.<*archetype mnemonic*>.<*version*>][*index*].<*property*>

or

[<*thread mnemonic*>.<*archetype mnemonic*>.<*version*>].<*property*>[*index*]

The *index* allows us to reference the individual property values, while the count property will indicate the maximum number of occurrences in existence.

Composite rules can be created, edited and compiled using the rules editor. This tool includes a browser that allows archetype references to be easily added to the code.

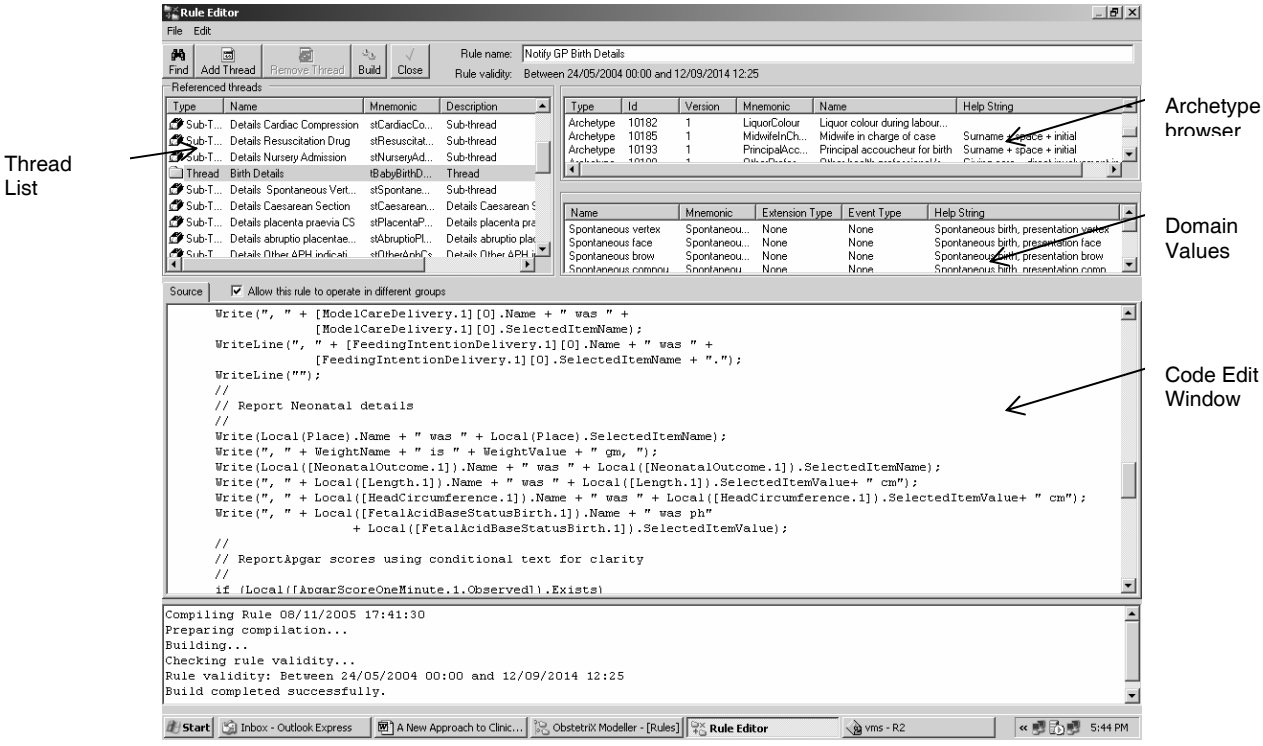


Fig 11 Archetype Editor and compiler

The rules capability used with the archetype-based system allows the expression of clinical knowledge to be conveniently modularised and embedded with the structure of the archetype.

Expert Users & Designer Competencies

In considering the concepts of application development using the two level model, we introduced a new type of user called a ‘domain expert’. In the previous section however we discussed a rules scripting language that involves logic expressions, object notation, control structures and local variables that would require skills generally provided by an experienced programmer. It has been demonstrated with the ObstetriX project that skilled clinical users (non-programmers) can build comprehensive information systems that include the type of simple rules logic shown above. It is

HIC 2006 Bridging the Digital Divide: Clinician, consumer and computer
Eds: Johanna Westbrook & Joanne Callen
Publisher: Health Informatics Society of Australia Ltd (HISA)
ISBN 0 9751013 7 4

inevitable however that more programming knowledge needs to be acquired to develop more complex rules logic.

The key advantage of the rules implementation described above is that the implicit coupling of information (via archetype object notation) to rules logic allows the user to focus on the logical expression of the rule rather than the programming needed to retrieve the data and manage the rule execution. This means that significantly less programming is required to express clinical rules and many powerful rules can be expressed in just a few lines of code.

Rules generation from data mining - closing the knowledge loop

The use of an archetype based system that is oriented towards clinical classification, has the potential to be developed into a valuable tool for improving clinical practice. The obstetric clinical repository in use in NSW contains many hundreds of thousands of cases where information has been recorded from first patient presentation through to delivery and discharge. This provides an opportunity to correlate presenting clinical details with actual birth outcomes in a process called data mining. Of particularly interest would be the analysis of certain adverse outcomes such as prematurity and other complicated births that put mother and/or baby at risk.

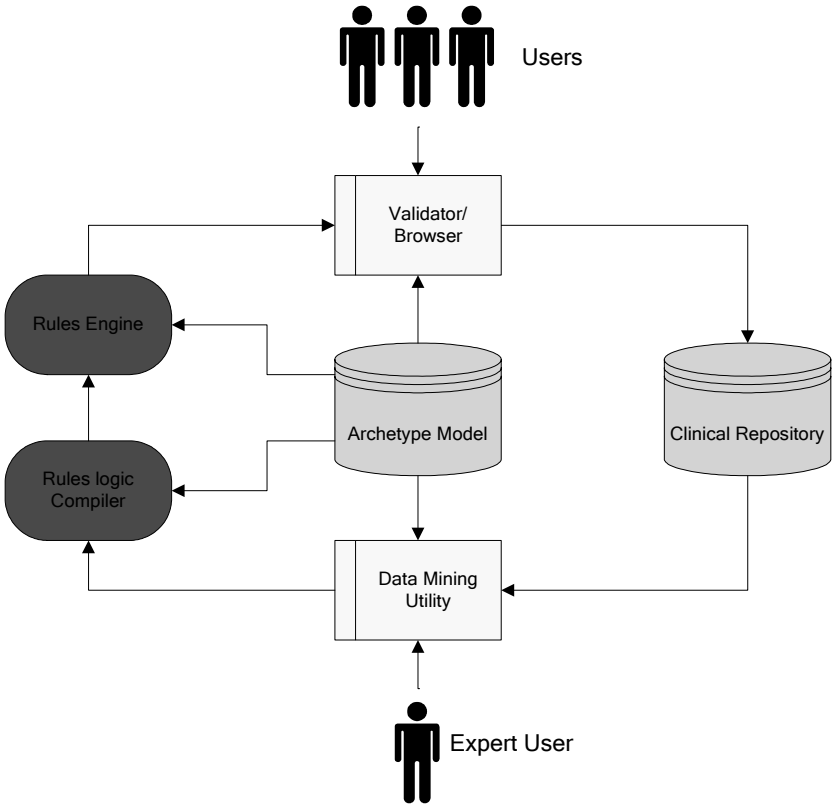


Figure 12 Data Mining Loop

The science of data mining can produce logic maps that can be expressed in terms of archetypes and therefore directly implemented as scripted rules. The application of these rules for a presenting patient would yield a quantitative measure of risk, and could result in interventions that would mitigate that risk. This suggests the possibility of implementing a process of continuous knowledge refinement as in Fig 12.

HIC 2006 Bridging the Digital Divide: Clinician, consumer and computer
Eds: Johanna Westbrook & Joanne Callen
Publisher: Health Informatics Society of Australia Ltd (HISA)
ISBN 0 9751013 7 4

Conclusions

This paper has outlined a practical alternate method of building a clinical information system based on two level data model architecture. The benefits of this approach have been confirmed by a major program of implementation of the ObstetriX system in birthing units in NSW Public Hospitals. The key benefits of this approach have been found to be:

- Dramatic savings in programming time can be achieved as domain experts rather than programmers can carry out development and maintenance of functional clinical components. The development of the generic components of the system can proceed independently as all 'subject' matter is stored in the clinical model.
- High level of user ownership and responsibility as the 'clinical model' is entirely created and maintained by clinical and expert users and is seen as a separate intellectual product to the generic programs.
- Easier for users to learn and operate. This is due to a single clinical model meta-structure for the Archetype so the user is presented with a consistent application interface. And because Archetypes are maximally normalised, they can be organised according to clinical workflow. The result is less training time, better user acceptance and data compliance.
- Complete data consistency through versioning. The use of Archetype versioning allow definitions to be changed without compromising historic data and for information to be viewed in the exact context in which it was recorded. Versioning is date and time enabled which allows changes to the clinical model to be made in a controlled way.
- The programming of business rules requires less effort and produces more stable and robust logic. The availability of the archetype object class in rules scripting allows the development of encapsulated rules that are easier to implement, maintain and verify. The developer is able to focus on the expression of the business rule rather than how data is retrieved and when the rule may be executed.
- Generic program components are more robust and less vulnerable to failure as they are not subject to constant modification. In the two level model architecture no subject matter stored in the database schema and the physical data schema is quite small, very stable and independent of archetype model changes.

This paper has shown the effectiveness of using two level archetype based systems in the clinical obstetric environment. This model for building Electronic Medical Record (EMR) and Electronic Health Record (EHR) systems has the potential for wider application in other clinical areas as well as any other area where there is a similar need for secure management of complex information.

Acknowledgements

I would like to thank the members of the NSW Obstet Consortium for their insight, faith and courage in supporting the development of both Obstet and ObstetriX. In particular for the strong support received from consortium chairman Dr Michael Nicholl (Senior Staff Specialist, Department of Obstetrics & Gynaecology, RNSH). As suggested elsewhere in this document, the 'two level' approach is only possible if there exists competent and dedicated 'subject matter experts'.

Thanks to midwives Kate Dyer and Helen Cook for their efforts in developing the first Obstet clinical model (that is still in use in many hospitals in NSW). And special thanks to Catriona Andronicos for stepping up to the many challenges of developing the 'new' ObstetriX model as well as managing the complexities of a most unconventional software project.

HIC 2006 Bridging the Digital Divide: Clinician, consumer and computer
Eds: Johanna Westbrook & Joanne Callen
Publisher: Health Informatics Society of Australia Ltd (HISA)
ISBN 0 9751013 7 4

This type of high-risk project, undertaken by a small software company, could not have proceeded without support from the NSW Department of Health. Thanks to Siobhan Finnerty and Peter Williams for their support for various stages of the project.

Finally thanks to Jongho Ju (senior software architect MHI) for his innovation and skill in the physical design and development of the ObstetriX system.

Bibliography

[1] Beale, Thomas 2000. Archetypes – Constraint Based Domain Models for Future Proof Information Systems. (Available at http://www.deepthought.com.au/)

[2] Ashton, David 1996. Data Modelling for Clinical Information Systems – Proceedings, Cognos User Conference, Terrigal June 1996.

[3] Dampney, Prof CNG 1998. Review of "Prototyping Clinical Information Systems (David Ashton, 1996)"

[4] Beale, Thomas 2002. A Shared Archetype and Template Language (Available at http://www.deepthought.com.au/)

[5] Albrecht A.J. 1979, 'Measuring Application Development Productivity' in Proc. IBM Applications Development Symposium. GUIDE Int and Share Inc., IBM Corp., Monterey, CA Oct 14-17,

[6] Date, C.J 1981. An Introduction to Database Systems Third Edition. Addison Wesley.

Gamma et al 1995. Design Patterns. Addison Wesley Longman.

Address for Correspondence

David Ashton
Managing Director
Meridian Health Informatics
51-57 Holt Street
Surry Hills NSW 2010
Phone: 9212 6055
Mobile: 0417 691197
Fax: 9211 4644
ashtond@meridianhi.com
www.meridianhi.com