The RIMBAA Technology Matrix

See http://www.ringholm.com/docs/03100_en.htm for the latest version of this document.
Author: René Spronk, Sr. consultant, Ringholm, and Ewout Kramer, manager R&D, Firely.
Document status: Draft, version 0.8 (2009-04-23)
Please send questions and comments to rene.spronk@ringholm.com.

Summary

An increasing number of software implementations in healthcare use a HL7 version 3 RIM based application architecture (RIMBAA). The subject of this whitepaper, the RIMBAA Technology Matrix, was created in order to be able to identify the architectural choices used by the designers of these applications in a technology independent fashion. This in order to determine a set of generic approaches and best practices related to the use of HL7 version 3 RIM-based models for purposes other than messaging.

1. Introduction

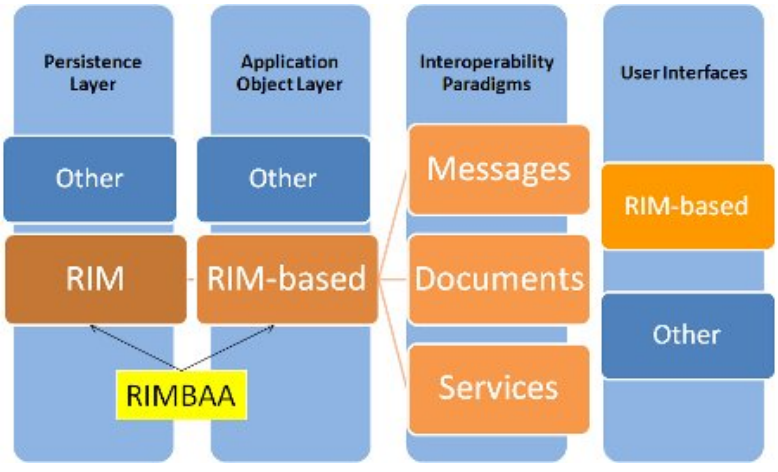
The focus of the HL7 standards, including the HL7 version 3 Reference Information Model (RIM), has traditionally been on the exchange of data. The internal architecture of software applications was regarded to be out of scope.

"*The RIM model is too good to be limited to messaging*" says Peter Hendler, one of the committee chairs of the HL7 RIM Based Application Architecture (RIMBAA) group. "*If someone were to show a programmer the RIM data model - and not give them any pre-information that it was created by an organization called HL7, and that the scope of HL7 is messaging - if you just put the data model in front of a programmer or an architect and say: please look at this model, and you ask them: what do you think this model is for?. Probably they'd look at the model and they'd say this is a very comprehensive model which could be used for making healthcare applications and could probably even be used to make persistence layers and relational databases.*"

Other implementers have come to the same conclusion (mostly because of exposure to the HL7 RIM in the context of messaging) , and are using RIM-based models as the basis for the development of applications. The RIMBAA technology matrix described in this whitepaper allows for a comparison between the architectural approaches used by these applications.

1.1 Rationale for the matrix

Several RIM-inspired real-world solutions related to the challenge of generating HL7 version 3 messages from persistent storage were presented during the April 2007 meeting of the Dutch HL7 version 3 Architecture SIG. Some applications use HL7 version 3 models to persist data, others at the application layer, and others as the basis for user interface generation. During the meeting one implementer presented an implementation that used the HL7 version 3 RIM throughout.

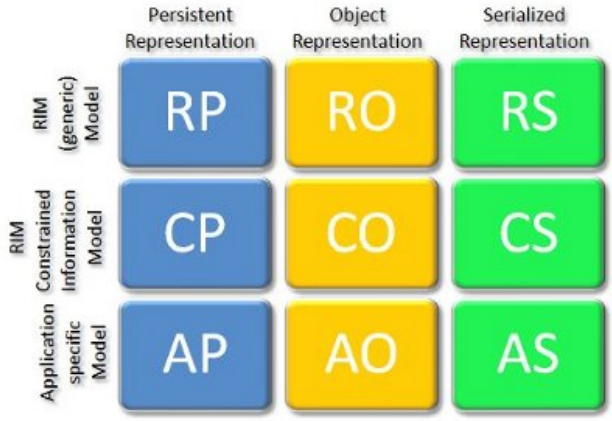


The solutions used different ways of persisting the data (e.g. a pre-existing database, or a newly developed fully RIM-based database) and totally different mapping technologies (e.g. .NET Serialization, SqlServer XML queries, JAXB generated Java code). This resulted in quite an extensive set of interesting approaches to achieve the same goal.

It was however quite hard to understand what technology was trying to solve which part of the overall problem. In order to determine a set of generic approaches and best practices a way had to be found to describe (and compare) the various approaches in a technology independent architectural fashion. Ewout Kramer, one of the members of the Dutch HL7v3 Architecture SIG, came up with the RIMBAA technology matrix, which allows us to do just that.

1.2 The RIMBAA technology matrix

The RIMBAA technology matrix contains all possible "roadmaps" to transition between the persistence layer and the serialized representation (e.g. XML messages) of the data.



The matrix is shown above. The top grid with 9 cells has two axis. One axis is the type of model (model dimension) you are using; the other is the current representation of the data (appearance dimension).

If an application is based on one of the left-hand cells one can travel across the cells of the matrix to one of the right-hand cells, whereby each transition from cell to cell leads to a choices in terms of technology. If we examine the existing RIMBAA implementations and determine what cells they use in terms of cell transition, we are effectively collecting a series of paths which have been (successfully) tried. Paths that have not been tried might be irrelevant, but could also indicate an alternative –as yet untried- architectural approach.

The way to use the matrix is to select a series of cells that an implementation uses to move from left-to-right (or vice versa). Most implementations support HL7 messages (the CS-cell, which stands for the XML-based, serialized, version of the message-level model). The persistence layer will often be implemented as a relational DBMS (the *P cells on the left hand side).

The approach could however differ between applications: existing application databases generally are not RIM-compliant but persistent application-specific relational models (the AP cell) whereas RIM-based solutions persist RIM-based models (the RP cell).

When it comes to the RP cell and relational databases there are three generic options:

Scope of the RO Cell

What's the scope of the RO cell? The RO cell is intended to be pretty close to a RIM-derived model. If an application uses an object model that was inspired on the RIM or a RIM-based model, but which requires mapping to a RIM-derived model (at the AO to RO interface) then the AO cell is more applicable than the RO cell.

RIMBAA or not?

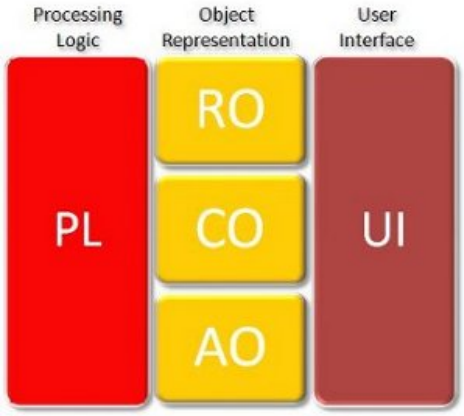
When is an application considered to be RIMBAA? This is the case if the application architecture uses at least one of the following cells: RP, RO, CP or CO.

- Table per Hierarchy. All the classes in a hierarchy (parents, children, grandchildren etc) are all persisted to one table. The table has to have columns for all possible members for all the children. In any given row, only the relevant columns are filled (so lots of nulls). One column is used as the "discriminator" that tells what type of class or subclass it is.
- Table per subclass. There is one table for the parent. All children that add something to this have side tables to contain the extra members. To get a child class persisted you always use at least two tables. The parent and the one that has the additional members for the subclass.
- Table per concrete class. Each class whether a parent or child gets it's very own table. No nulls, but lots of tables have similar columns because whatever the classes share, it is duplicated in each table.

Transitions between cells could be supported using one of several alternative approaches. The cell transitions won't be the same for all applications - a partial overlap in paths is however likely, which means that the experiences and best practices associated with that path can be shared between implementers.

1.3 The third dimension of the matrix

The Technology Matrix has a third dimension: the middle column (the *O cells) is one and the same; it covers the ways how RIM-objects are related to the Processing Logic (PL) and the User Interface (UI).



On the assumption that one uses the RO cell (of the Technology Matrix) Procesing Logic could be done in multiple ways. The extremes are:

- Content driven logic: based on whatever is contained in the data instance.
- Context driven logic: based on the context of the data, e.g. based on knowledge that the data conforms to a set of constrained RIM models (e.g. an Interaction, a MessageType or a Template).

In general, the CS-CO cell transition is mostly context driven, and the RS-RO cell transition is mostly content driven.

A few implementations are creating solutions related to the user-interface: the generation of dynamic user interfaces or forms. The goal is the semi-automatic generation of a UI (mostly based on a CIM), with controls that are suitable for the (flavoured) data types being used, with support for the appropriate binding of value sets, and validation of the data that is entered according to the appopriate templates and OCL statements.

When asked what one would 'normally' use to generated a user interface, Ewout responds with: "*This is an area were I haven't seen a lot of experiences yet.*" and continues with "*I'd say that we first transform it into an object model, and then from the object model you go into the user interface. I know that some people are doing translations directly from the message into a user interface. How one would do that - receiving data and validating it, is quite difficult. I think the object model plays an important role here.*"

UI's may be based on the RO, the CO or the AO cell as defined in the Technology Matrix. There are two (extreme) options:

- Run time: Dynamically (at run time) generate the UI based on whatever is contained in the data instance, or the CIM/LIM-definitions that data instance is known to conform to. (An example of the latter approach are MIF to XForm transformations).
- Development time: Define the UI first, and bind elements in the UI to RO/CO/AO classes/attributes.

An interview with Ewout Kramer about various aspects of the RIMBAA technology matrix can be seen here:

2. Conclusions and recommendations

The technology matrix is all about a reference model driven approach to application architectures. In general, not just when discussing the HL7 RIM, it is beneficial to use one and the same semantic (reference) model for messaging, in-memory application objects as well as at the persistence layer. This ensures the lowest degree of semantic loss due to mappings between data models.

The OpenEHR architecture (which mainly focuses on the RO cell) regards the consistent use of one single reference model as an essential requirement for implementations: "*The meaning of information must be preserved from this point down through layers of business logic to its representation in the persistence (data storage) layer. If this is not the case, information entered by a user will not safely reach the part of the system which allows it to be shared with other users or other systems, or even the same user at a later time.*. The same page criticizes messaging standards for their limited focus on serialized messages (the CS cell) without specifying the details of the application stack (the RO cell).

The video below contains an interview with two software developers about RIMBAA and the Technology Matrix. Both work for large HIS vendors, and have to deal with legacy applications and legacy data.

The HL7 RIMBAA Working Group has created a Java based Reference Implementation. This can be used to gain experience with this type of architecture. Whilst it is too early to draw conclusions (we're still in the process of gathering RIMBAA-technology-matrix based descriptions of implementations), some observations can be made related to current HL7-RIMBAA implementations:

- Mapping of application internal data models to message models is possible if one only has to support a couple of messages, or if the messages themselves are of relatively low complexity. Mapping is too cumbersome if one has to support a large amount of message structures, or if the messages contain complex clinical data. An increasing number of application architects are seeing the value of having consistent semantic models both for the application stack as well as for messaging.
- Schema based code generation (the RO or CO cell), as well as Object relational mappers (the RP or CP cell) are off the shelf software that are used in a sizable number of applications.

3. See also

- Blog post: HL7 creates a RIM Based Application Architecture (RIMBAA) group
- Interview (on Youtube) with Peter Hendler about the HL7 RIMBAA group (known up to June 2008 as the HL7 Java SIG)
- Blog post: HL7 UK AGM and RIMBAA
- (the RIMBAA page on) HL7 Wiki (anti-spam u: wiki, pw: wikiwiki)

About Ringholm bv

Ringholm bv is a group of European experts in the field of messaging standards and systems integration in healthcare IT. We provide the industry's most advanced training courses and consulting on healthcare information exchange standards.
See http://www.ringholm.com or call +31 33 7 630 636 for additional information.