Proceedings of the 9th IASTED International Conference
Software Engineering and Applications (SEA 2008)
November 16 - 18, 2008 Orlando, Florida, USA

THE REFERENT TRACKING SYSTEM AS A PEER 2 PEER APPLICATION

Shahid MANZOOR, M.Sc.[1], Werner CEUSTERS, M.D.[1], Ron RUDNICKI, M.A.[1],
Robert ARP, Ph.D.[2]
[1]New York State Center of Excellence in Bioinformatics & Life Sciences,
University at Buffalo, NY, USA
[2]National Center for Biomedical Ontology, University at Buffalo, NY, USA

ABSTRACT
A Referent Tracking System (RTS) is an application that manages a database containing data that represent real world entities, where each such entity is assigned a singular and globally unique ID. The application stores references to: (1) individual entities that exist in reality, (2) the relationships that obtain between these entities, (3) the universals instantiated by these entities and (4) terms from terminologies used in their description. In this paper, we describe the architecture of an RTS application, built by using the Peer 2 Peer (P2P) paradigm, that enables the data to be shared over distributed Peers running at geographically different locations.

KEY WORDS
Medical Informatics, Knowledge-based Systems and Peer-2-Peer Application.

1. Introduction

Referent Tracking (RT) was introduced in 2005 to avoid ambiguities that arise while referring to entities in Electronic Health Record (EHR) statements despite the use of terms drawn from standard terminologies such as SNOMED CT.[1] For example, in statements originating from two distinct encounters with the same patient, the use of the SNOMED CT code 71620000 (*fracture of femur*) by itself does not resolve the ambiguity whether reference is being made to one fracture or to two distinct fractures of the same type. RT resolves such ambiguities by assigning singular and globally unique identifiers - called Instance Unique Identifiers (IUIs) - to the particular entities referred to in reality. Thus if the patient suffers (or has suffered) from two distinct fractures, then these fractures are referred to by two different IUIs, even if the two fractures are of exactly the same type, and, as a consequence, are described by means of the same SNOMED CT code. If, in contrast, only one single fracture is being documented during distinct encounters, then only one IUI would be used. RT thus advocates an all-encompassing use of IUIs, not only assigning them to individual patients, but also to their body parts, disorders, diagnoses, and the physicians who treat them.

Data in RT consist of statements in the form of tuples (see Table 1), where each statement represents a fact – modulo the occurrence of errors for instance introduced because of a false belief – about at least one particular in reality.[2-4] Consider, for example, the following tuples:

A_1: <IUI-1, IUI-2, 2/8/2008>
$PtoN_1$: <IUI-1, 2/8/08, name, John, IUI-2, 2/8/08, IUI-3>
$PtoC_1$ <IUI-1, 2/8/08, SNOMED CT, 116154003, IUI-2, 2/8/08>.

Table 1: RT Tuples

Representation Name	***Attributes Set***
Description	
A-tuple	$< IUI_a, IUI_p, t_{ap}>$
Act of assignment of *IUIp* to a particular at time t_{ap} by the particular referred to by author IUI_a	
D-tuple	$< IUI_d, IUI_A, t_d, E, C, S >$
A D-tuple is inserted (1) to resolve mistakes in RTS, and (2) whenever a new tuple other than D is inserted in the RTS. The particular referred to by IUI_d registers the particular referred to by IUI_A (the IUI for the corresponding A-tuple) at time t_d. E is either the symbol 'I' (for insertion) or any of the error type symbols (insert ref to my OIC-2007 paper). C is the reason for inserting the A-tuple. S is a list of IUIs denoting the tuples, if any, that replace the retired one.	
PtoP-tuple	$<IUI_a, t_a, r, IUI_o, P, t_r>$
The particular referred to by IUI_a asserts at time t_a that the relationship *r* from ontology IUI_o obtains between the particulars referred to in the set of *IUIs P* at time t_r.	
PtoU-tuple	$<IUI_a, t_a, inst, IUI_o, IUI_p, UUI, t_r>$
The particular referred to by author IUI_a asserts at time t_a that the particular referred to by IUI_p instantiates *inst* relation from ontology IUI_o with the universal *UUI* at time t_r.	
PtoC-tuple	$<IUI_a\ t_a, IUI_c, IUI_p, CUI, t_r>$
The particular referred to by IUI_a asserts at time t_a that it is annotated by concept code associated with *CUI* from terminology system IUI_c at t_r.	
PtoU(-) -tuple	$<IUI_a, t_a, r, IUI_o, IUI_p, UUI, t_r>$
The particular referred to by author IUI_a asserts at time t_a that the relation *r* of ontology IUI_o does not obtain at time t_r between the particular referred to by IUI_p and any of the instances of the class *UUI* at time t_r.	
PtoN	$< IUI_a, t_a, nt_i, n_i, IUI_p, t_r, IUI_c>$
The particular referred to by IUI_a asserts at time t_a that n_i is the name of the nametype nt_j used by IUI_c to denate the particular referred to by IUI_p at t_r.	

These tuples express that the author whose IUI is IUI-1 asserts on Feb 8, 2008 that the entity referred to by IUI-2 is a patient (as indicated by the SNOMED CT code 116154003 to which is associated the term 'patient') whose name 'John' is used in IUI-3 (which stands for the USA) and that these descriptions are (believed to be) true on the same date.

In 2007 we built a Referent Tracking System (RTS) prototype which manages RT tuples in a database.[5] The RTS was developed as a client-server application, where an RTS client could be either an EHR application calling the services directly, or a middleware application communicating with the RTS server on behalf of the EHR application.[6] The RTS server provides its interface via web services that, for instance, insert a new RT tuple in the database (e.g. 'createParticularRepresentation', 'createPtoN', and 'createPtoP'), or which search tuples (e.g. getParticularRepresentation', 'getPtoN', and 'getPtoP').

2. Objectives

Because the key idea in RT is making reference to entities in reality by means of singular and globally unique identifiers, the optimal set up would be one in which only one RTS would be used worldwide. More realistically, however, is the adoption of the RT paradigm in a step-wise fashion: each healthcare related institution will first install its own RTS, and afterwards connect them in expanding networks.

To support this evolution, we developed a new version of the RTS which is built upon Peer to Peer (P2P) technology, enabling data sharing in such a way that a search query can be executed concurrently over distributed RTS servers (peers). In an RTS P2P network a client thus sends a query to an RTS server which besides executing the query itself can forward it to other connected RTS servers for subsequent execution. Each peer then collects the results and sends them to the requesting peer. Finally, the RTS server who received the initial request returns the aggregated results to the client.

Furthermore, an RTS P2P application is capable of database load sharing over multiple RTS server peers such that the network behaves as a singular database. This capability is useful in cases where a very large database cannot be hosted on a single machine, for instance because of computational limits.

3. Material & Methods

P2P architectures can be implemented by means of any programming language which has network programming capabilities. Some P2P applications, for example, are implemented by using web services. However, such an approach requires the announcement of each peer and its services to the other peers by means of a centralized server such as a UDDI repository.[7] But if the immediate adoption of one global RTS is unlikely, so also is the adoption of one worldwide UDDI server.

JXTA is another platform, introduced recently, that is specifically designed to build P2P applications and that consists of a set of protocols that are independent of any programming language.[8] It has built-in capabilities for discovering a new peer in a network, for authenticating users, and for ensuring secure communication. The JXTA community has implemented JXTA protocols as an API for three environments: Java standard edition, C/C++, and Java Micro Edition. We have adopted the JXTA API for the Java standard edition.

JXTA protocols are centered around the architectural construct of a 'group' which stands for a collection of peers using a common, agreed-upon set of services. A 'group' is identified by a Globally Unique ID (GUID). All of the JXTA services, e.g. group membership, Input/Output pipes (streams), messaging, etc., are accessible in the context of this group. Each system in a JXTA network is required to join a group to access its services. The group membership can be authenticated by a user ID and password. A peer can only join a group when it knows the group GUID, user ID, and password. A JXTA network can have more than one group, and a peer can be a member of more than one group.

4. Results

4.1 RTS P2P Application

Our application design is a mix of client-server and P2P programming models. The RTS P2P network consists of several RTS peers of three distinct types: (1) RTS Server Peers that only execute the queries (as a central server) received from other network peers, (2) Proxy Peers which function as clients of Server Peers and which provide interfaces as a Java API (by implementing JXTA protocols) to RTS clients, and (3) ServerProxy Peers that act as a combination of Server Peers and Proxy Peers by first accepting and executing query requests from other Proxy Peers and then forwarding the queries to other Server Peers. The RTS clients, typically EHR clients, do not need to know about the JXTA protocols: they just have to call the API methods of their Proxy Peer to send a query, which forwards the query to the connected RTS Server Peers.

An example of the RTS network is shown in **Figure 1**, where two health care institutes, A and B, are running their own RTS peers. The peers with dotted background are installed in such a way that they are not directly known outside their corresponding health care institute's environment. In Health Institute A, the three Server Peers (with dotted background) are alike in all respects and implement the objective of distributing a very large database load. When an EHR client (in health institute A) sends a search query to the Proxy Peer, it forwards the

query to the three Server Peers which concurrently execute the query and return the results to the Proxy Peer that finally sends the results to the EHR client. For public access to each health care institute's data, the separate peers (with gray background) are installed. The idea of separating the peer advertisement in local (within a health institute) and public (outside the health institute) contexts is to build a security layer. The peers which are known locally provide full access to the local database, and the peers which are known publicly provide very restricted access to the database (e.g., they might, for instance, allow only searches over PtoN tuples).

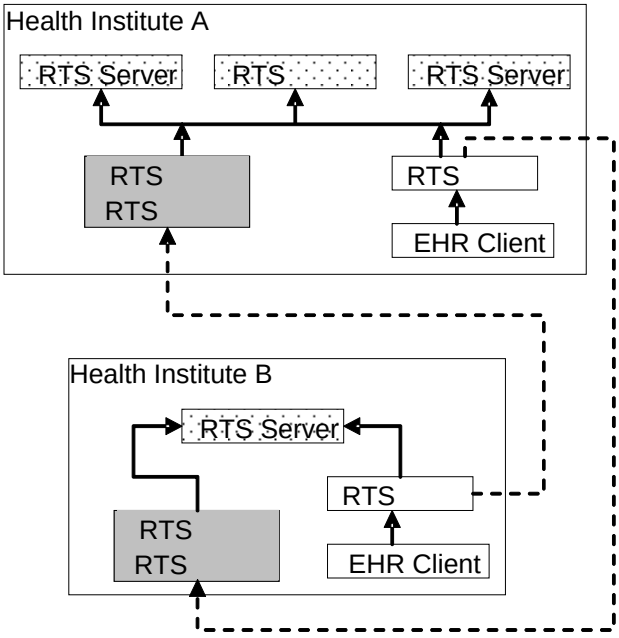

Figure 1: An Example of an RTS P2P Network

In **Figure 1**, the scope of the peers (i.e., local and public) is delimited with the help of the JXTA group construct. Each institute can form its own local group whose membership (shown by solid lines) is not known outside their corresponding health care institute environment, and which protects against unauthorized access to the peers in the group. There is also a public group of which both health care institutes are aware (whose membership is shown by dotted connection lines).

4.2 RTS P2P Application Architecture

In **Figure 2**, our P2P application architecture is schematized to be composed of several component layers, where arrows indicate the direction of the information flow. The Client Side layer contains the RTS Client (these could be third party EHR applications or middleware components), which sends a query to a Proxy Peer in the network layer that forwards the request to the appropriate RTS server in the network. During the execution of the query, the RTS server calls the services of the RTS core API to retrieve the results from the RDBMS databases that constitute the data source layer.

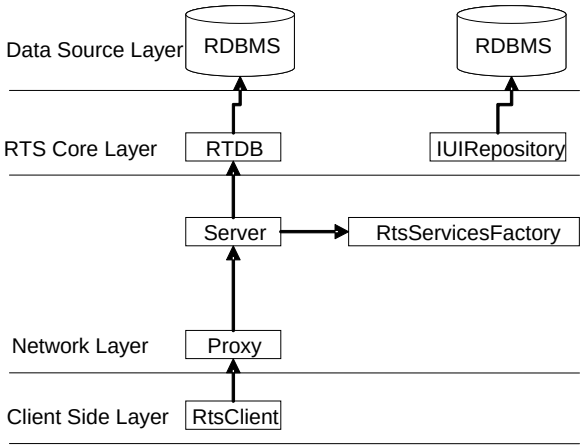

Figure 2: RTS P2P Architecture

4.3 RTS Core Layer

The RTS Core layer implements the business logic of RT, namely, the insertion and retrieval of RT tuples in a database. We have used the existing RTS Alpha 1 source code[9] to build this layer, and made modifications such that we have split the RTS database into two database applications: the IUIRepository and the RTDB. The IUIRepository database manages the statements about the assignment of IUIs to particulars, and provides a central repository of IUIs to the RTS. The RTDB is a database of statements representing the detailed information about particulars, examples being '#IUI-1 instantiates the universal Person' and '#IUI-1 has the name "John".'

The IUIRepository and RTDB components are implemented as Java APIs. The IUIRepository contains services to search particular representations and to insert new ones in its corresponding RDBMS. Similarly, the RTDB components provide API get methods (e.g., getPtoN, getPtoP etc.) to search and create methods (e.g., createPtoN and createPtoP, etc.) to insert tuples in its database.

The IUIRepository and RTDB components are implemented independently of any specific RDBMS (e.g., MYSQL, HSQL). RDBMS support is controlled by an RDBMS specific driver component. Currently, we have implemented drivers for MYSQL and HSQL.

4.4 RTS Network Layer

The network layer provides the communication services to send or receive messages over a network. In this layer, the Server and Proxy components use internally the JXTA protocol for communication, and run within the scope of a group. A Proxy Peer component can communicate with a Server Peer component only when both components are members of the same group. Furthermore, if a peer in a health institute provides server services for two groups, then it has to run the Server Peer component for each group.

RTS Server:

The server component provides central query execution services to a Proxy Peer functioning as client. The server is implemented in a way similar to the Services Oriented Architecture[10] (based on an idea similar to that of web services) in which a set of services (similar to remote procedures) are provided as a query mechanism. The XML language is used to send both query and results between peers. Implementing the query mechanism by using XML avoids making changes in the server and proxy components as new services are introduced.

Listing 1 shows an example of the createPtoN service which allows inserting PtoN tuples in the database. The *Name* element inside the *RtsService* element would hold the name of the service, and the elements inside the *Params* element would contain the parameters of the service. For the sake of simplicity, only two parameters ('iuip' stands for the IUI of the particular for which statement PtoN is inserted and 'ta' stands for the time at which the PtoN statement is inserted) are shown in this example.

In our architecture, it is not required that all server peer installations provide the same set of services. Services in a server are published via *RtsServicesFactory*, a Java interface which returns the list of service handlers where each service handler is responsible for the execution of a specific service. Each service handler is implemented as a Java class which has two Java methods: getServiceName() and handlService(queryXML).

```
<RtsQuery>
  <RtsService>
    <Name>createPtoN</Name>
     <params>
         <iuip>IUI-50</iuip>
         <tr>1201890219266</tr>
         …
     </params>
  </RtsService>
</RtsQuery>
```

Listing 1: An Example of an RTS Service Query

The method getServiceName() returns the name of the service, e.g., createPtoN, for which this handler is implemented. The RtsServer component calls this method to match the service name in the query (which is sent by a client for execution). If the query name is matched with getServiceName, then the server calls the handleService method of this handler.

The handlService(queryXML) method handles the execution of a service and returns the results in the form of XML to the server. Then, the server sends the XML, including its header information (which is used only for internal purposes between server and clients), to the client who sent the query.

To publish new services in a server, only the server configuration file is modified to tell the server about the availability of the new services implemented as a java class (implements the java RtsServicesInferace interface).

RTS Proxy:

RTS Proxy is the client side implementation of the RTS server that provides interfaces to RTS clients which do not have knowledge about the JXTA. Currently, RTS Proxy is implemented as a Java API, but it can also be implemented using web services for those clients that are implemented on something other than the Java platform.

The output of the proxy client, when querying multiple servers in a group, is based on the idea of *streaming* such that it outputs a result as soon as it receives it from a server.

Just after building a successful connection to a server, a Proxy Peer requests a list of services from the Server Peer (e.g., insertPtoU, getPtoU, getPtoN, etc). The Proxy Peer uses this information to forward the RTS client query to the appropriate servers which handle the query. For example, in **Figure 1** one server alone (out of the other servers with dotted background running in Health Institute A) handles the createPtoN service and, should a client request the Proxy Peer to execute the createPtoN service, the Proxy forwards the service call to that RTS server which handles the createPtoN service.

We have implemented the Proxy component in such a way that it need not be changed if a server announces a new service. Since the service calling mechanism uses XML as a data format, the proxy component provides a utility method to build a service query in XML.

5. Discussion

The RTS P2P application described here is built by using three programming paradigm techniques: Client/Server, Service Oriented Architecture (SOA), and P2P. In the Client/Server model, EHR applications (as a client) running at different departments in a health institute access the services of a RTS system (running as a central server). The SOA model gives us the opportunity to use parameterized remote procedures as a query mechanism, where a parameter represents query criteria; for example, the procedure call 'findPatientByName("John")' finds all patients whose names are matched with the parameter "John." The P2P architecture provides the opportunity to access the data services of the RTS systems running at geographically different locations.

The JXTA P2P technique resolves scalability of our application in terms of:

- Data Magnitude: A very large database can be distributed over multiple machines, which could reduce hardware costs.

- Performance: Concurrent execution of a query over multiple machines reduces the query execution time.
- Security: The JXTA group concept is a very good security tool because services running in the context of a JXTA group can only be accessed by those Peers who are authorized to join the group. In this way, a group of health care institutes could form a consortium to share a common set of data access services (associated with a JXTA group) from their RTS Server Peers. This design could be used to prevent anyone outside the consortium from accessing these services. Our application allows building such consortiums within a network.

At the other hand, distributing data geographically has significant potential issues such as data transmission costs, current servers and networks performance, amount of data stored on each peer, available free disk space, possible administrative constraints (such as restrictions for remote accesses during high-peak local loading periods), reducing the number of unnecessary queries and avoid network congestion, and so forth. These issues depend on the following factors.

- Data Size in Transmission: any peer-2-peer application has to bear the data transmission cost which by the user is primarily experienced in the form of response time as data flows. The greater the data size flows in a transmission from one peer to another peer over a network, the larger the response time may be experienced. This is because network bandwidth allows a specific data size to be transmitted at a time. In our application a peer could send data ranging from a few kilobytes to several megabytes. This is in the first place determined by the policies that a health institute adheres to in order to provide access to its data to clients, in addition to load reducing techniques. Larger size data can be sent into multiple smaller size data packets or the data size in transmission can be reduced by providing services that can only return very specific information about a particular patient at a time. For example, one can allow to ask about a specific fracture or tumor of a patient, or a blood pressure values over restricted time intervals, rather than ‘give me all that is available’.

- Topology Model: There are different topological network models available for p2p applications, each with distinct modes of data centralization. Most p2p applications are developed for file sharing systems. In pure p2p systems such as Gnutella [11] and Freenet [12] all peers have equal roles in data searching or downloading. In a hybrid model such as in Napster [13], a central server provides search capabilities, while downloading occurs at individual peers. The hybrid model is more efficient than the pure p2p model as important data (such as which files exist on what peers) can be indexed on the central server with the result that searches can be performed very efficiently: after a peer executed a search on the server to retrieve a location for certain data, it makes direct connection with the peer who has the data for immediate download. However, if the central server is inaccessible, then the shared data are not visible to peers. A model that solves this problem, is the Superpeer model as implemented in Kazaa [14] which is a mix of both the pure and hybrid model[15]. A Superpeer provides centralized services (as in the hybrid model) to a subset of peers. Client peers send queries to their Superpeers and get results from them. A Superpeer also routes queries to other Superpeers on behalf of its clients, and then collect results from the other Superpeers to return them to its clients.

  Most jurisdictions have regulations which do not allow health institutes to publish their patient data on a public central server. Therefore we have followed in our implementation the pure p2p network model. In this model, each participating health institute has to form a consortium with the other health institutes with which it wants to share data. In our model, each peer knows the other peers in a consortium so that it can make direct connections with the other peers for querying data. Furthermore, upon reception of a query, each public peer only forwards the query in its internal private network and does not forward the query to the other public peers. Even under this scenario, data may only be shared without patient consent if they are rendered anonymous for instance to search for finding similar cases with respect to symptomatology, disease, treatment and so forth.

- Machine Performance: because our implementation is such that peers make direct connection with the other peers, failure or slowing down of one peer does not cause failure of the entire network. In the worst case, if a public peer of a health institute fails then the data of that particular institute will not be available to others. This problem can be handled by introducing redundant public servers so that in case the public server fails the second redundant server will take over the job. The machine performance is also affected by the sort of hardware configuration and what RDMBS is used. The RTS performance was individually tested in 2007 for 1.3 million RT tuples on a desktop machine (Core 2 duo E6400 processor, 1 GB RAM, 60 GB hard disk and 5.1 MySQL database).[5] With the latter settings, a peer conforming to our implementation would perform

well. Improving the hardware configuration, a peer would perform certainly better.

- Network Performance: Overall network performance does not depend on one individual connection amongst two peers; if a network connection fails between two peers or if there is congestion at one edge, then that does not slow down the entire network.

6. Conclusion

An RTS application contains data about the entities (patients, disorders, body parts, diagnoses, and so forth) that are referred to in EHR statements. In contrast to prevailing approaches, RTS data are structured in a way that mirrors the nature of these entities and their relationships in reality, rather than how they are classified in terminologies or reported upon in documents[16]. RT is therefore capable of providing additional detail that cannot be exchanged by means of, for instance, HL7's Clinical Document Architecture since the latter has no mechanisms to specify what the data are *about*[17]. The use of our P2P application helps thus to achieve the objective of sharing this sort of detailed patient data across multiple health care institutes in a secure way. This, then, would make it possible to further advancements in building computational systems concerned with decision-making, patient data analysis, and data interoperability.

With the use of Service Oriented Architecture, efforts are being made to produce solutions for the data interoperability between EHR applications. Such approaches could leverage our P2P model so as to build the most robust and dynamic solutions.

References

1. W. Ceusters and B. Smith: Referent Tracking in Electronic Healthcare Records. *Connecting Medical Informatics and Bio-Informatics. Medical Informatics Europe 2005*Amsterdam, IOS Press, 2005, p. pp. 71-76
2. W. Ceusters and B. Smith: Strategies for Referent Tracking in Electronic Health Records, *Journal of Biomedical Informatics* 2006, *39*(3):362-378
3. W. Ceusters, P. Elkin and B. Smith: Referent Tracking: The Problem of Negative Findings. *Studies in Health Technology and Informatics. Ubiquity: Technologies for Better Health in Aging Societies - Proceedings of MIE2006*Amsterdam, IOS Press, 2006, p. pp. 741-746
4. W. Ceusters: Dealing with Mistakes in a Referent Tracking System. *Proceedings of Ontology for the Intelligence Community 2007*Columbia MA, 2007, p. pp. 5-8
5. S. Manzoor, W. Ceusters and R. Rudnicki: Implementation of a Referent Tracking System, *International Journal of Healthcare Information Systems and Informatics* 2007, *2*(4):41-58
6. S. Manzoor, W. Ceusters and R. Rudnicki: A Middleware Approach to Integerate Referent Tracking in EHR Systems. *Proceeding of AMIA Annual Symposium*2007, p.
7. OASIS UDDI Specification Technical Committee. *UDDI OASIS Standard*. [cited 2008 Febuary 26, 2008]; Available from: http://www.oasis-open.org/specs/index.php#uddiv3.0.2.
8. J. Community. *JXTA*. [cited 2008 Febuary 26, 2008]; Available from: https://jxta.dev.java.net/.
9. S. Manzoor. *Referent Tracking System*. [cited 2008 September, 10, 2008]; Available from: http://sourceforge.net/projects/rtsystem.
10. Prakash M. Nadkarni and R. A. Miller: Service-oriented Architecture in Medical Software: Promises and Perils, *J Am Med Inform Assoc* 2007, *14*(2):244-246
11. *The Gnutella Protocol Specification v0.4*. [cited 2008 October 08]; Available from: http://www.securitytechnet.com/resource/hot-topic/p2p/GnutellaProtocol04.pdf.
12. *The Free Network Project*. [cited 2008 October 08]; Available from: http://freenetproject.org/.
13. Napster LLC *Napster*. [cited 2008 October 08]; Available from: http://www.napster.com.
14. Sharman Networks. *Kazaa*. [cited; Available from: http://www.kazaa.com/us/index.htm.
15. B. Yang and H. Garcia-Molina: Designing a Super-peer Network. *In Proceedings of the 19th International Conference on Data Engineering (ICDE)*Bangalore, India, 2003, p.
16. R. Rudnicki, W. Ceusters, S. Manzoor and B. Smith: What Particulars are Referred ti in EHR Data? A Case Study in Integerating Referent Tracking into an Electronic Health Record Application. *Proceeding of AMIA Annual Symposium*2007, p.
17. B. Smith and W. Ceusters: HL7 RIM: An Incoherent Standard. *Studies in Health Technology and Informatics. Ubiquity: Technologies for Better Health in Aging Societies - Proceedings of MIE2006*Amsterdam, IOS Press, 2006, p. pp. 133-138