5-18-2013

AGGREGATE DATA MODELING STYLE

Vladan Jovanovic
Georgia Southern University, vladan@georgiasouthern.edu

Steven Benson
Georgia Southern University, steven_r_benson@georgiasouthern.edu

AGGREGATE DATA MODELING STYLE

Vladan Jovanovic
Georgia Southern University
vladan@georgiasouthern.edu

Steven Benson
Georgia Southern University
steven_r_benson@georgiasouthern.edu

ABSTRACT

The paper presents our progress in defining an aggregate data modeling style using new operators with the standard data modeling language Idef1X. By defining a style, using an established notation, the proposed allows reuse of existing data modeling expertize. The style is intended to support data modeling for so called aggregate-oriented NoSQL databases.

Keywords

Data Modeling, Idef1X, NoSQL, Aggregate Data Model, Aggregate Data Modeling Style, data modeling operators

INTRODUCTION

Any system, seen as a set of entities together with their relationships and properties of those entities and their relationships, when observed, can be represented with data values (recorded in a database). For the convenience of database design a number of entity relationship models and languages had been used. We selected Idef1X, a standardized (FIPS 1993) data modeling language, as a basis for defining a style to address modeling for contemporary NoSQL databases. The Idef1x have fewer concepts than UML and is directly translatable into relational model by tools, such as CA Erwin. Before elaborating a new modeling style we will first introduce the Idef1X concepts and a working classification of NoSQL databases.

Of the Idef1X visual elements (see Figure 1) we use Entities (independent as rectangles and dependent as rounded rectangles), Relationships (dotted lines for regular 1:1 or 1:M relationships, and a solid lines for identifying 1:M relationships), a cardinality of a relationship (the many i.e. M= 0,1,…, side is implied on the dot side of a relationship, and one is implied on the opposite), and optionality (diamond on the one side of a relationship). The advanced concept of generalization is illustrated showing two variants (intentionally complete and incomplete subtyping) both with mutually exclusive subtypes. Notice that we are not using unspecific i.e. M:M relationships that Idef1X otherwise supports for conceptual modeling, as those need to be resolved (replaced by intersection entities before .

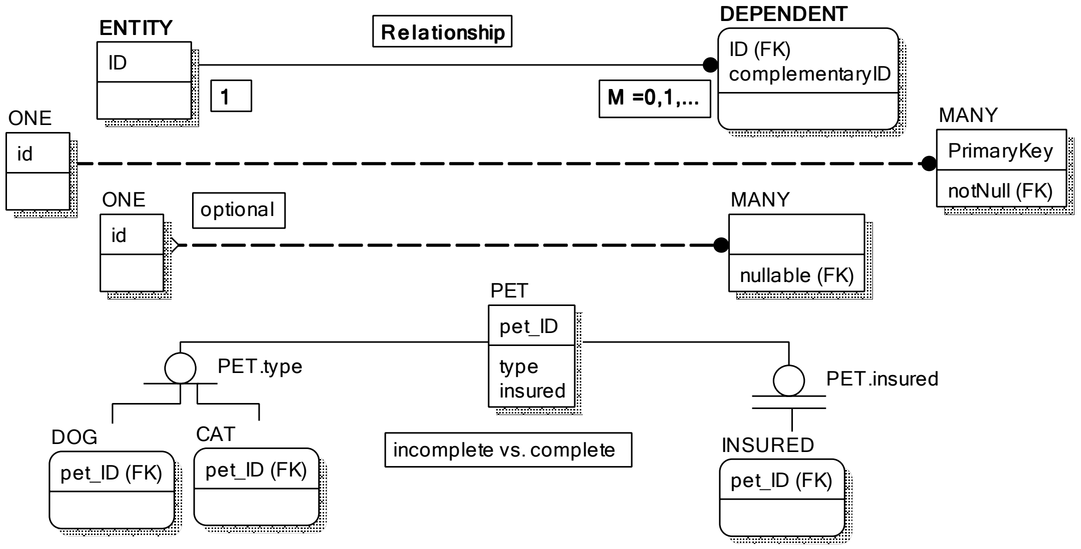

Figure 1. Visualization of Idef1x concepts

The practice of reifying M:M relationships, see Figure 2, is necessary for making specific logical models, i.e. models that can be automatically translated into relational schema, and is also a pre-requisite for the proposed aggregate data modeling style.

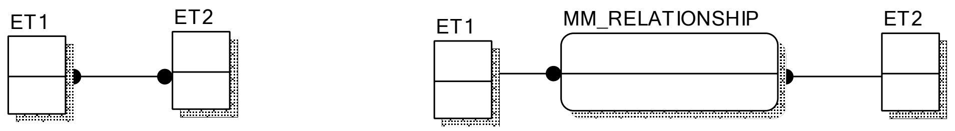

Figure 2. Reification of a M:M relationship into an intersection entity

Now let us address a classification of NoSQL databases in order to narrow the scope for the aggregate modeling style. The term NoSQL, and related DB engine products, evolved from a “no SQL”, via “not only SQL”, to a “new SQL” according to (Stonebraker 2009, and Stonebraker 2012), but the term is not to be treated as an acronym (Sadalage and Fowler 2013). There is no definitive classification of NoSQL databases but uses by (Cattell 2010; Leavitt 2010; Stonebraker 2012; Clarence, Arawinth and Shreeharsha 2012) shows a substantial convergence. The (Sadalge and Fowler 2013) designated as aggregate-based only the following NoSQL database categories: **Key-value** stores (Dynamo, Redis, Scalaris), **Column-wise** stores (Hbase, MonetDb), and closely related **Column-family** stores (BigTable, HyperTable, Casandra), and **Document** stores **(**CouchDB, MongoDB, SimpleDB). This clearly leaves NoSQL Graph DB and Text DB out of our scope. The products listed as examples of NoSQL databases differ in their implementation using specific file systems and collectively have no conceptual or even logical model that differs from the physical implementation model (as specific file system). Nevertheless they share the property of being able to store aggregated data clusters. The term *aggregate* here follows the notion of aggregate as a rich structure of closely related data that makes sense to be stored as a unit because it is almost exclusively accessed as a unit. Modeling such aggregates is not a new phenomenon; we can trace it to the Semantics Object Model (Kroenke 1995) and Domain-Driven Design (DDD) (Evans 2004). A review of sources influencing our work must include (Voughn 2013) elaborating on design of aggregates in DDD. The need for deliberate data modeling with NoSQL DB was recognized in (Schram, and Anderson 2012). An informal overview of data modeling for NoSQL was provided by (Katsov 2012); Esposito 2011), and in considerable depth in the (Sadalage and Fowler 2013). Nevertheless, our observation is that a better defined approach (to data modeling for NoSQL databases) is needed, specifically tapping into a reservoir of traditional data modeling expertize (for relational databases). This observation both motivated and directed our investigation leading to formulation of an aggregate data modeling style.

AGGREGATE DATA MODELING STYLE

A modeling style can be defined by patterns and constraints that produce models of recognizable form. The aggregated data model is, formally, a forest of independent aggregates (i.e. trees). Of the three aspects of modeling data i.e. structure (syntax), semantics (substance i.e. meaning), and pragmatics (convenience), we will mainly address the structural aspect of achieving the aggregate form. Our recommendation regarding semantics, is to use traditional data modeling to develop conceptual data models first (with detail analysis of entities, attributes, and the relationships required for satisfying functional requirements) and then apply new operators to recast models into the aggregate form for NoSQL implementations. Specific domain analysis is necessary also to recognize aggregates (according to responsibilities for updates i.e. transactions) as well as references (to nomenclatures maintained in separate systems and/or applications). Pragmatic considerations, are mainly related to non-functional requirements (performance, scalability, etc.), and can be relegated to physical design under a specific NoSQL database engine to guide trade off evaluation of logical design alternatives (as alternative aggregates). We will completely and formally define an aggregate data modeling style with a procedure based on the use of specific operators specified by structural applicability patterns. The pre-requisite, needed to prime data models is a single dependency pattern designating entity with responsibility for updates for each of dependent entities. The data modelers need to use the single dependency pattern deliberately (Figure 3) while making a conceptual models (after internalizing this pattern, as it may require some time to get used to it) or as a first step in transforming a traditional conceptual data model into an aggregate form.

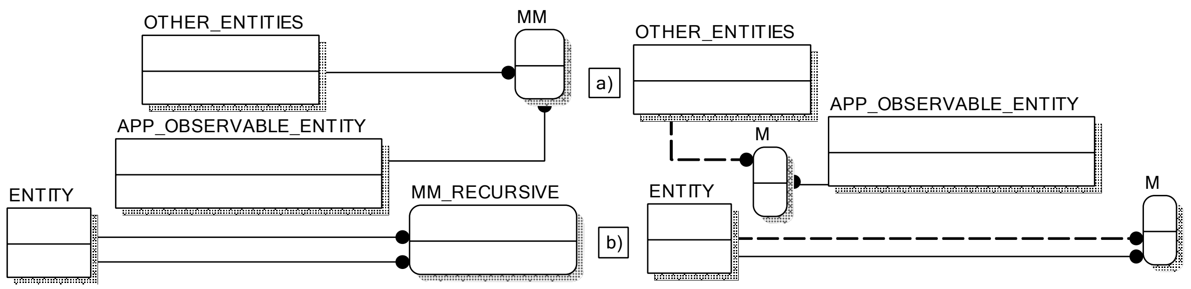

Figure 3. Single Dependency Pattern variants

The reduction of data models with only single dependency patterns to aggregated form of a set of trees (possibly collapsed) can be achieved by eliminating relationships either by 'disconnecting' entities (after copying referenced identifiers in a manner of foreign keys), or by 'nesting' entities similarly to de-normalization pre-join patterns (Ligstone, Torey, and Nadeau 2007). An entity must be designated as a root for each aggregate (future data cluster with a unique identity) to serve as its focal point during analysis and for eventual NoSQL physical implementation. In order to designate root entities and to create their reference entities (containing only root entity identifier necessary to support relationships with other aggregates), we propose a ROOT operator. To minimize clutter we may show reference entities only as necessary (as in Figure 5), as they can be implied whenever a ROOT is used. Our specification of an aggregate data modeling style proposes only two additional operators, REFI and EMBED, to indicate relationships to be 'eliminated' during the data model reduction into an aggregate form. The patterns a, b, and c for EMBED are shown on Figure 4, and patterns d, e, f, g, and h, for REFI on Figure 6. The REFI is absolutely required to formally reduce, cut a network (graph) to trees, and EMBED is necessary to reduce model size to fewer entities by explicit collapsing (nesting) of entities, as well as to conform to specific NoSQL DB file system for physical implementation.

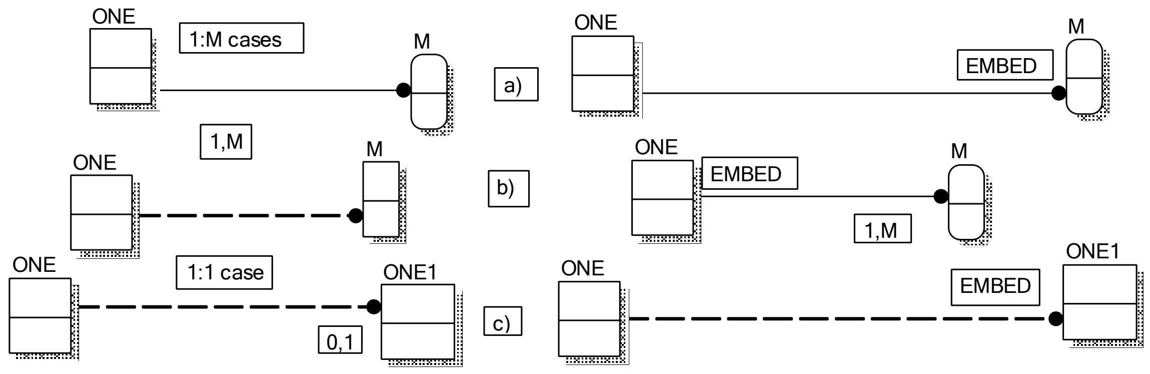

Figure 4. Patterns for EMBED

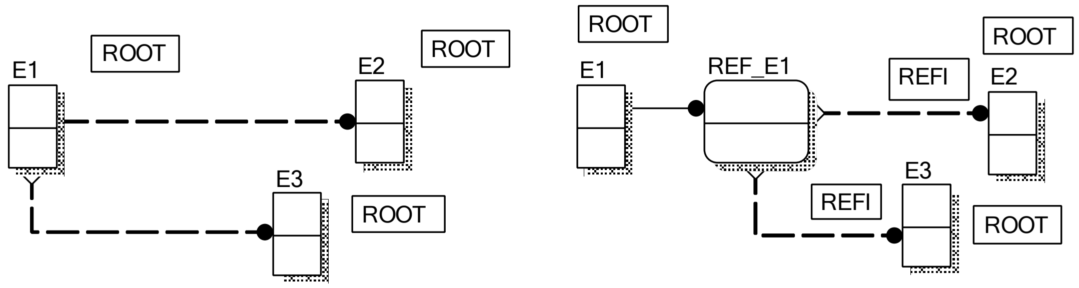

Figure 5. Relating aggregates only via (explicit) reference identity

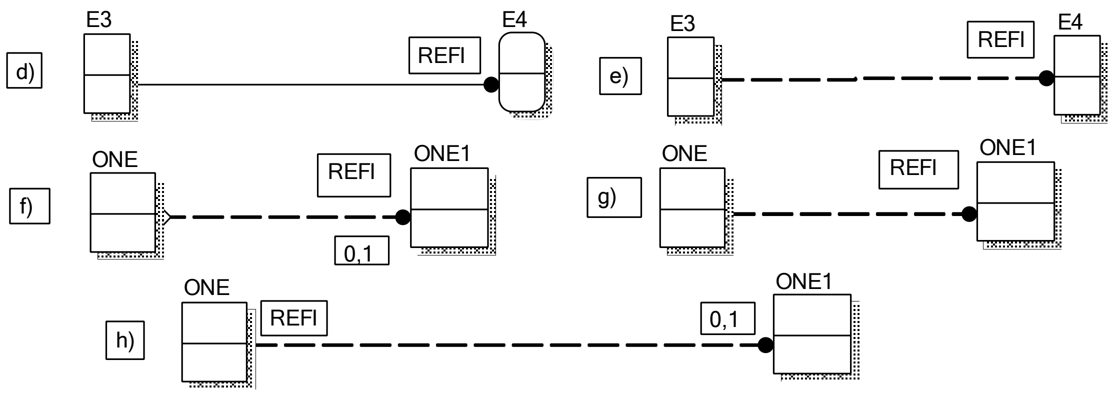

Figure 6. REFI Patterns

The use and positioning of EMBED operator is illustrated on Figure 4. The EMBED is to be placed on the side of the entity to be embedded (typically a dependent one). The EMBED is also applicable in designating subtypes to be physically included into a variant super type, use of this particular pattern, a special case of pattern (c) i.e. a 1:1 relationship of a supertype with its subtypes, is illustrated in Figure 9. The REFI operator is positioned with the entity where only a reference value will be copied into (effectively realizing a relationship by severing the connection to it but without eliminating referenced entity) see Figure 6. The examples of use of the ROOT operator are shown by Figures 9 and 10.

AN ILLUSTRATIVE EXAMPLE

To provide a demonstration, one traditional data model (Figure 8), treated as a conceptual model, is presented in the aggregate style (Figure 9) for a Medical Record system decoupled from semantically related system of applications, via reference entities shown in green. The context systems comprising of drug approval and registration, procedure and device registration, generic drug substitutions, application for advisory information on multiple drug interactions etc. are not shown here to simplify presentation and focus only on creating an aggregate data model. The Figure 9 shows a logical model with one aggregate and Figure 10 an alternative data model, typical in SOA, with five aggregates. The subtype of REGISTERED_PROFESSIONAL, on Figure 8, is shown in yellow only to clearly indicate a requirement that can be postponed or added. We use symbol X to indicate relationships to externally managed reference data, as points of connection with external systems.

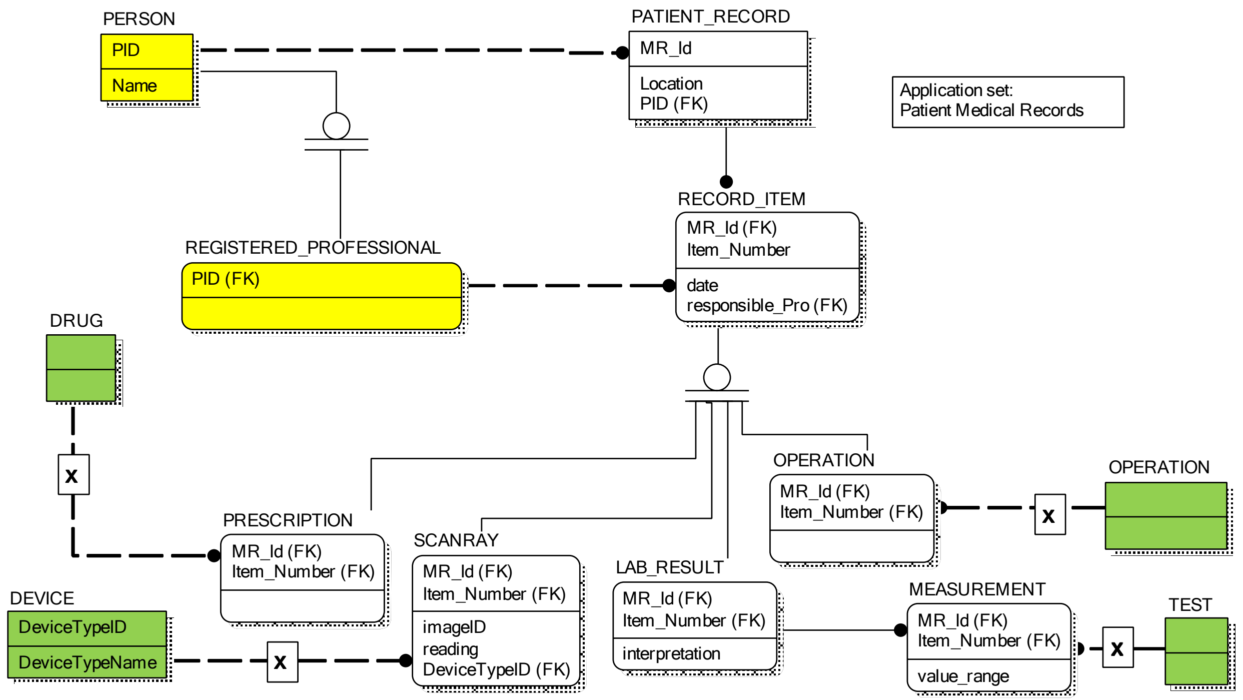

Figure 8. Conceptual data model for decoupled application set- Patient Medical Records

The making of an aggregate data model is not an automatic process, a modeler must substantively decide scope of transactions and delineate aggregates based on a real work design (i.e. assignment of responsibilities to users, typically captured via workflow tasks and/or use cases), mark separate aggregates by their ROOT entities, and decide which relationships need to be 'eliminated' by REFI and which 'collapsed' by EMBED. Nevertheless, we can explain a mechanism assuring the aggregate form of a model formally. The reduction of a data model to selected aggregates (clusters around ROOT entities) requires: a) severing references (where each REFI is applied individually), and b) recursive application of EMBED culminating into a ROOT. An application of EMBED is kind of pre-join de-normalization applied from the bottom up. In the example on Figure 9 we have first the nesting of the REGISTERED_PROFESSIONAL data into a PERSON. Second, we have a case of pushing a parent into the child, effectively a redundant copying of a PERSON data into the PATIENT_RECORD. On the separate branch MEASUREMENTS are first embedded into the LAB_RESULT and then all four subtypes embedded (nested, pre-joined) into RECORD_ITEM. Finally RECORD_ITEM is nested into PATIENT_RECORD forming a single cluster super entity i.e. the aggregate PATIENT_RECORD. The Figure 10 shows an alternative logical data model, using different choice of ROOT entities, typical with smaller service oriented applications (where each ROOT is shown in yellow for emphasize).

One implication of aggregates for NoSQL implementations is really important to point out, namely that each transaction instance will need only a single entity cluster instance allowing freedom in mapping all the database data to various locations in a distributed environment without substantial performance penalties. The only (logically necessary) data to be replicated are value references to external systems, and the references to separate aggregates (if any) in the same application/system. Those are the only two cases for the 'eventual consistency' mechanism, while 'immediate consistency' in transactions is not causing a problem in a distributed environment (Voughn 2013) whenever all the aggregates are fully separated.

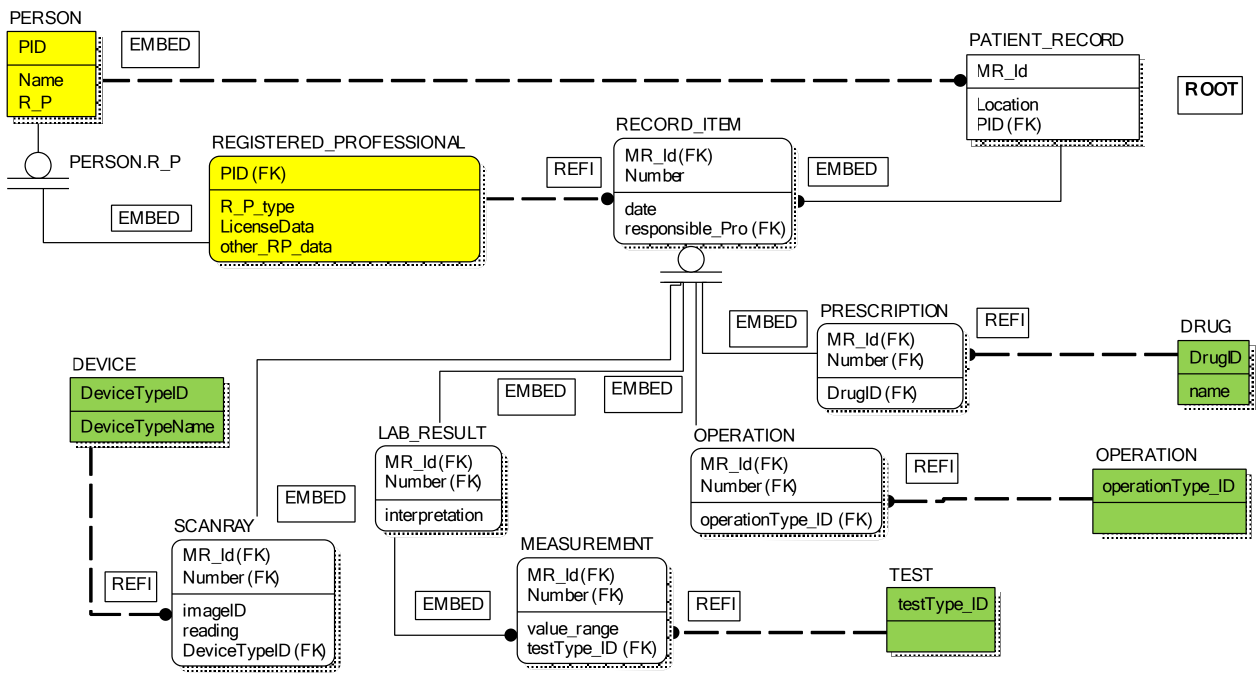

Figure 9. Logical data model reduced to a single aggregate

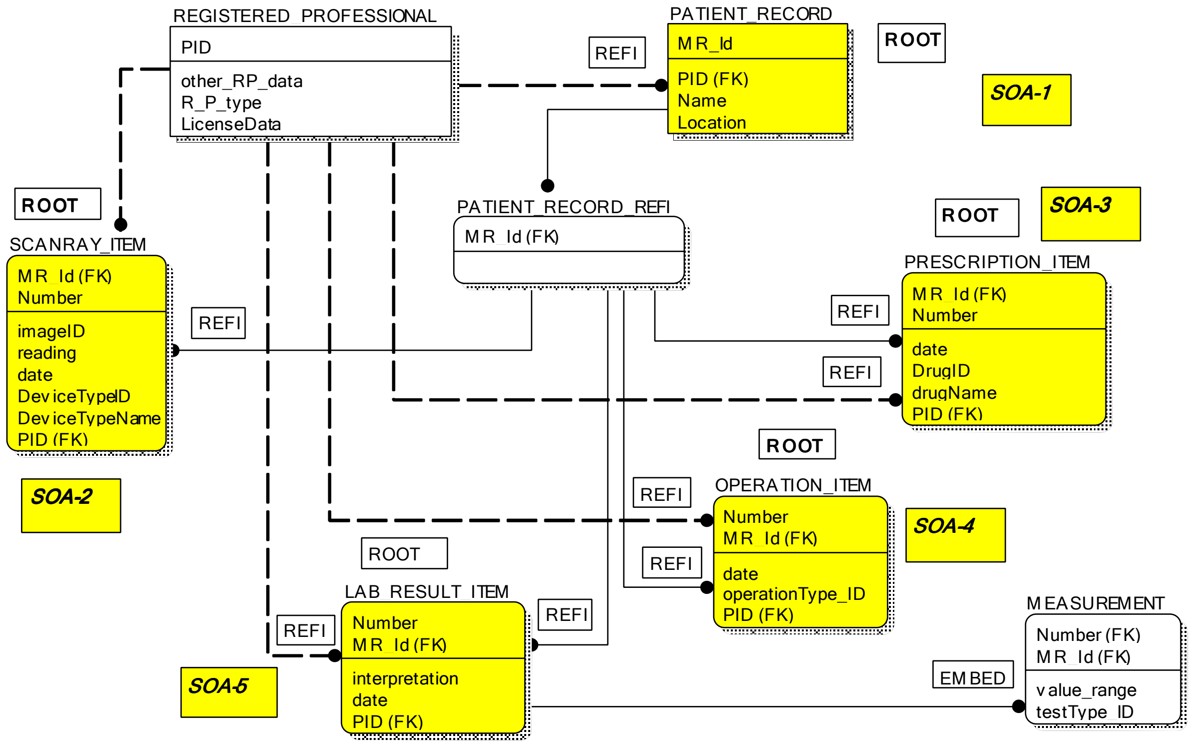

Figure 10. Logical data model with five aggregates in a SOA environment

CONCLUSION

We are aware that the NoSQL databases opened a new non-trivial problem area for data modelers. For most analysts as data modelers concerned with application data requirements that are relegating bulk of implementation to NoSQL programmers, an aggregate data model represents a missing link between traditional data models and aggregate-oriented NoSQL DB physical file systems. One of the key points of this paper is that work on conceptual and logical modeling for aggregate NoSQL DB can prepare foundation for automation of future physical design as well as make aggregate data models ready for custom coding. All three aggregate operators can be treated as instructions for formal transformations to at least some default physical implementation (with selected aggregate NoSQL database), in a manner similar to how contemporary case tools generate SQL DDL code in transforming traditional data models into relational model. Research under our supervision as well as CA's Erwin and Oracle's SQL Developer Data Modeler (among other case tools) are actively pursuing formal transformations to a select NoSQL engines. Physical data modeling obviously represent an involved problem due to varieties of non-standardized and rapidly evolving NoSQL engines. In principle for each NoSQL selection, supported data types and other available options are to be planned for and all formal transformations defined before full automation become feasible.

Our work on logical data modeling with aggregate style already provides medium for trade off analysis (as different aggregates can be selected i.e. modeled and resulting model variants as preliminary physical models compared). In the service oriented architecture (SOA), where applications are organized as tiny services reducing data model to really small and independent pieces is a pivoting goal. The aggregate modeling style is well suited to SOA as it can reduce traditional data models into very small aggregated data models systematically, see the illustration of the service based transaction driven aggregate model form (Figure 9) and its SOA based alternative shown in Figure 10.

Directions for future work include: a) in depth exploration of opportunities for standardizing physical data modeling in the aggregate data modeling style for diverse NoSQL database engines, and b) defining model refactoring and transformations for code generation in a style of (Ambler and Sadalage 2007) again for different NoSQL DB engines. In addition, there is a relatively minor issue that can have significant pragmatic benefits, namely, refining a role of color to support aggregate data modeling style. Also of interest is providing important/involved examples of models and complete implementations for the explicit purpose in using them in the education of future modelers and DB designers. A recommendation for the use of colors is an issue where progress can be made fast, as it is limited only by experience of developers and availability of resources for experimentation, and some results in the context of traditional data models (equivalent to 3rd or ideally 5th normal forms) had been presented before.

REFERENCES

1. Ambler, S, and Sadalage, P. (2006) Refactoring Databases, Addison Wesley,
2. Cattell, R. (2010) Scalable SQL and NoSQL Data Stores, SIGMOD Record, V39, N4, December 2010,
3. Clarence J., Aravindh S., and Shreeharsha, A. (2012). Comparative study of the New Generation, Agile, Scalable, High Performance NOSQL Databases, Int. J. of Computer Applications V 48, N20, June
4. Esposito, D. (2011) Objects and the Art of Data Modeling, MSDN Magazine, October,
5. Evans, E. (2004), Domain-Driven Design, Addison-Wesley,
6. FIPS 184 (1993) Integration Definition for Information Modeling (IDEF1X),
7. Katsov, I. (2012) NoSQL Data Modeling Techniques, *http://highlyscalable.wordpress.com/2012/03/01/*
8. Kroenke, D. (1995) Database Processing 5rd edition, Prentice Hall,
9. Leavitt, N (2010) Will NoSQL Databases Live Up to Their Promises? IEEE Computer V43, N2, pp 12-14,
10. Ligstone, S., Torey, T., and Nadeau, T. (2007), Physical Database Design, Morgan Kaufmann Publishers,
11. Sadalage, P. J. and Fowler, M., (2013) NoSQL Distilled: a brief guide to the emerging world of polyglot persistence, Addison-Wesley Upper Saddle River, NJ,
12. Schram, A. and Anderson, K. M. (2012) MySQL to NoSQL: data modeling challenges in supporting scalability, *Proceedings of the 3rd annual conference on Systems, programming, and applications, ACM*, New York, 191-202.
13. Stonebraker M., (2009). The NoSQL Discussion has nothing to do with SQL , blog@ACM, November 2009,
14. Stonebraker, M. (2012) New Opportunities for New SQL, Comm. ACM V55, N11,
15. Voughn, V. (2013) Implementing Domain Driven Design, Addison Wesley.