A standardized drug nomenclature links systems that use different vocabularies, so the patient gets what the doctor ordered.

Simon Liu, Wei Ma, Robin Moore, Vikraman Ganesan, and Stuart Nelson

RxNorm: Prescription for Electronic Drug Information Exchange

Today's commercial drug information systems follow a variety of naming conventions. For example, cooperating hospitals might use different systems, and find their data incompatible. Even within a hospital, there might be one system for ordering, another for inventory management, and still another for recording dose adjustments or checking drug interactions. A smooth electronic exchange of the information in these systems—not only between organizations but even within a single organization—is crucial in assuring patient safety. This exchange requires a standardized nomenclature.

To meet this need, the National Library of Medicine (NLM) created RxNorm, a standardized nomenclature for clinical drugs that is one of a suite of standards designated for use in US federal government systems for the electronic exchange of clinical health information. The goal of RxNorm is to let various systems using different drug nomenclatures share and exchange data efficiently. It provides a way to link standard clinical drug names to many of the drug vocabularies commonly used in pharmacy management and drug interaction software, including those of First DataBank, Micromedex, Medi-Span, and Multum. By linking these vocabularies, RxNorm can mediate messages between systems that use different vocabularies. These linkages ease interoperability among the computerized systems that record or process data dealing with clinical drugs.

RxNorm will eventually cover all prescription medications approved for use in the United States. In addition, when authoritative information is available about prescription medications from other countries and over-the-counter medications, RxNorm will add and cover these drugs as well.

DATA REPRESENTATION

RxNorm is based on a model developed at the NLM in consultation with the Health Level 7 vocabulary technical committee and with the Veterans Administration. It expresses what a clinician might order for a patient, and the type of order a pharmacy might recieve. In the RxNorm nomenclature, a clinical drug's name is a semantic normal form (SNF) that reflects its active ingredients, strength, and form (the physical form in which the drug is administered or specified to be administered in a prescription or order). When any of these elements varies, we add a new RxNorm drug name to the nomenclature. Thus, RxNorm includes a name for every strength and dose of every available combination of clinically significant ingredients.

Concepts

RxNorm organizes data by concept. A concept is a collection of names identical in meaning at a specified level of abstraction; using concepts, RxNorm can recognize strings of characters from disparate sources as the same thing. For example, *Accuneb, 0.042% inhalation solution* and *Albuterol 0.417*

1520-9202/05/$20.00 © 2005 IEEE Published by the IEEE Computer Society

Figure 1. Semantic network example.

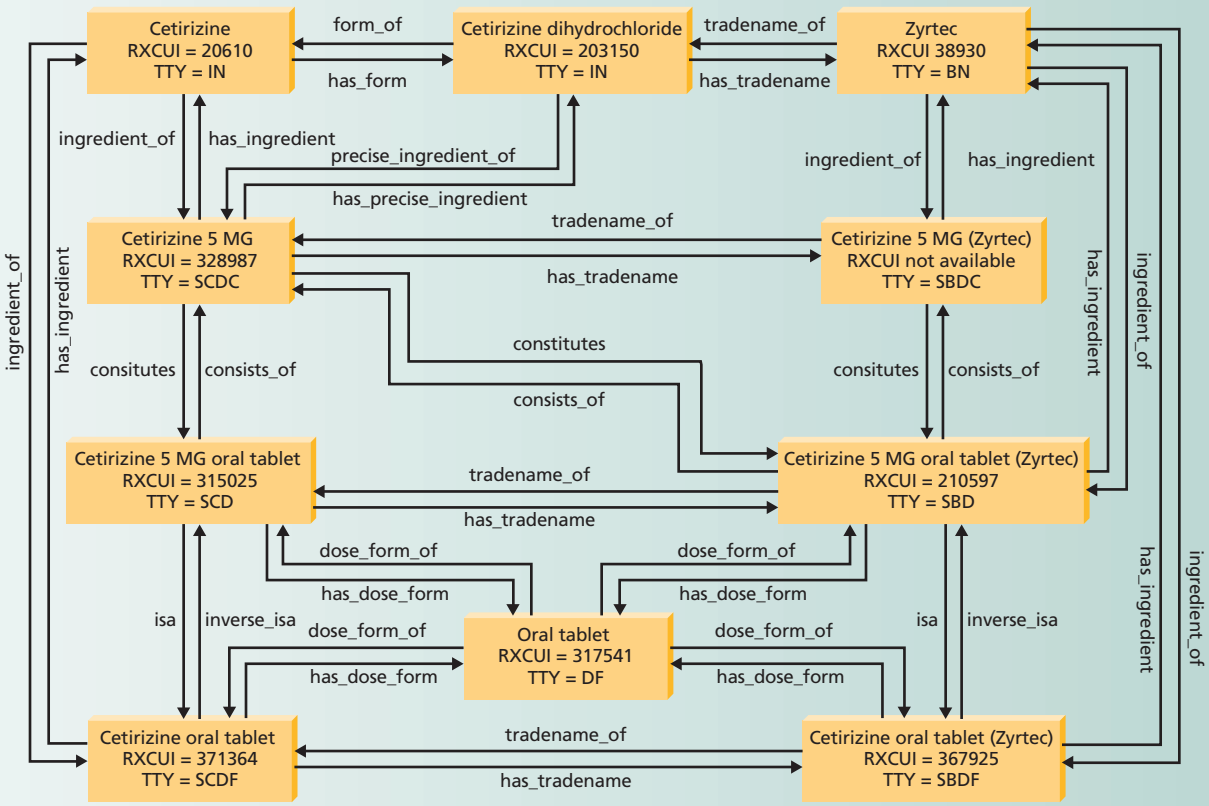
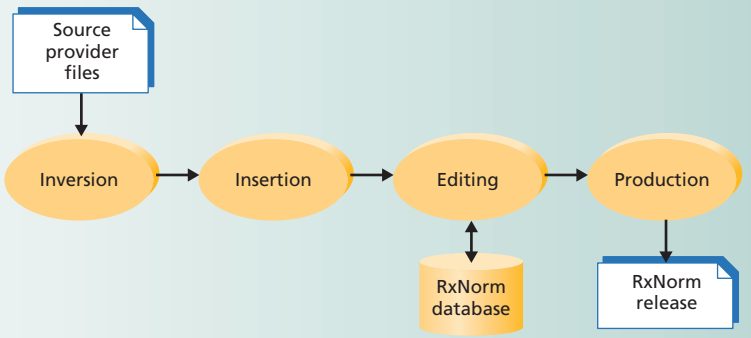


Figure 2. RxNorm data life cycle.



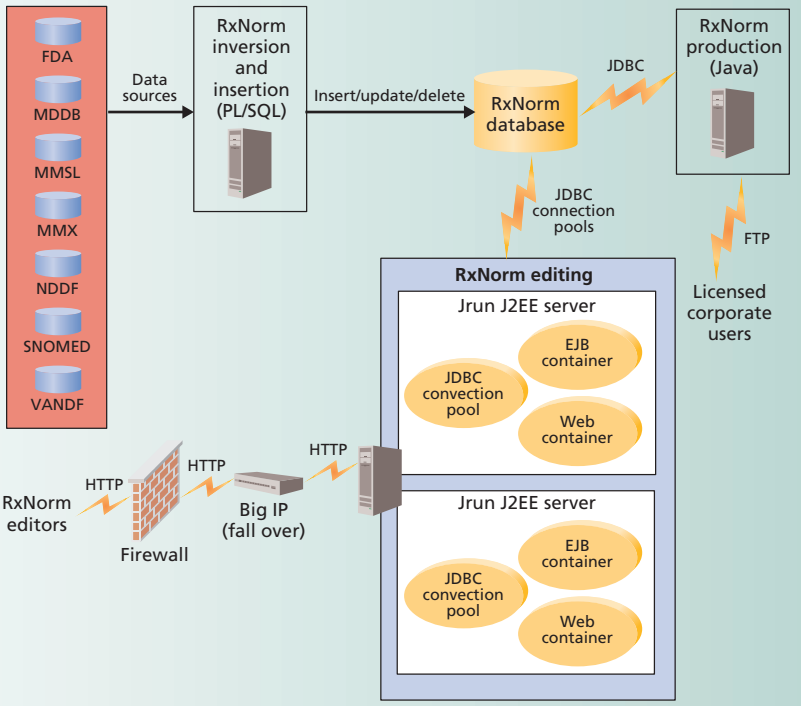
MG/ML Inhalant Solution [Accuneb] name the same concept. RxNorm designates the second of these the preferred form of the name and assigns this concept an RxNorm concept unique identifier (RXCUI)—in this case, 352051. This RXCUI always designates the same concept, no matter the form of the name. Drugs whose names map to the same RXCUI are taken to be the same drug—identical in ingredients, strengths, and dose forms. Conversely, drugs that differ in any of these particulars are conceptually distinct and have different RXCUIs.

RxNorm follows a standard format in naming clinical drugs, and it uses the RxNorm naming conventions to normalize drugs named differently in various other vocabularies. The normalized form of a clinical drug name includes several elements, each of which can be identified by the value of the term type (TTY). Major TTY values include,

- IN—ingredient. This is a compound or moiety that gives the drug its distinctive clinical properties. Examples: *Fluoxetine, Insulin,* and *Isophane.*
- DF—dose form. Example: *Oral Solution.*
- SCDC—semantic clinical drug component. This repre-

sents the ingredient plus strength. Example: *Fluoxetine 4 MG/ML.*
- SCDF—semantic clinical drug form. This represents the ingredient plus dose form. Example: *Fluoxetine Oral Solution.*
- SCD—semantic clinical drug. This represents the ingredient plus strength and dose form. Example: *Fluoxetine 4 MG/ML Oral Solution.*
- BN—brand name. This is a proprietary name for a family of products containing a specific active ingredient. Example: *Prozac.*
- SBDC—semantic branded drug component. This represents the branded ingredient plus strength. Example: *Prozac 4 MG/ML.*
- SBDF—semantic branded drug form. This represents the branded ingredient plus dose form. Example: *Prozac Oral Solution.*
- SBD—semantic branded drug. This represents the ingredient, strength, and dose form, plus brand name. Example: *Fluoxetine 4 MG/ML Oral Solution [Prozac].*

Figure 3. RxNorm system architecture.



Relationships

Various relationships exist among RxNorm concepts, and, together, concepts and relationships form a semantic network. Relationships between concepts in RxNorm are reciprocal. For example, a clinical drug consists of components, and the components constitute the clinical drug. This means that a concept with a TTY field value of SCD bears the relationship "consists_of" to certain other concepts with a TTY value of SCDC, and each of these, in turn, bears the relationship "constitutes" to the first concept.

RxNorm uses the following relationships:

- constitutes/consists_of,
- contains/contained_in,
- dose_form_of/has_dose_form,
- form_of/has_form,
- ingredient_of/has_ingredient,
- isa/inverse_isa,
- precise_ingredient_of/has_precise_ingredient, and
- tradename_of/has_tradename.

Figure 1 illustrates a semantic network with associated concepts and their relationships.

RxNORM DATA LIFE CYCLE

Figure 2 diagrams the life cycle of RxNorm data, which consists of four major steps. The first step, *inversion*, is the process of converting source provider files in their native format to the RxNorm common format. Inversion begins with a set of files that come from a source provider. Those files run through a series of processes to produce a Rich Release Format (RRF) file that RxNorm can use. The second step, *insertion*, is the process of loading inversion RRF data files into the RxNorm database.

Third, *editing* is the process of resolving conflicts between a source's view and the RxNorm view. It is a human editor's task to incorporate all the different source provider views into a homogenous single representation. Editors receive work lists that are collections of concepts. An editor typically looks at one or more RxNorm concepts at a time to resolve the issues causing the conflicts. Once a concept is consistent, an editor approves it.

Fourth, *production,* an automated process, involves extracting releasable data from the RxNorm database and then producing and validating the release. Validated releases, in RRF, are then available for download by the user community.

RxNORM SYSTEM ARCHITECTURE

Figure 3 diagrams the RxNorm system architecture, which

Figure 4. Software component architecture.

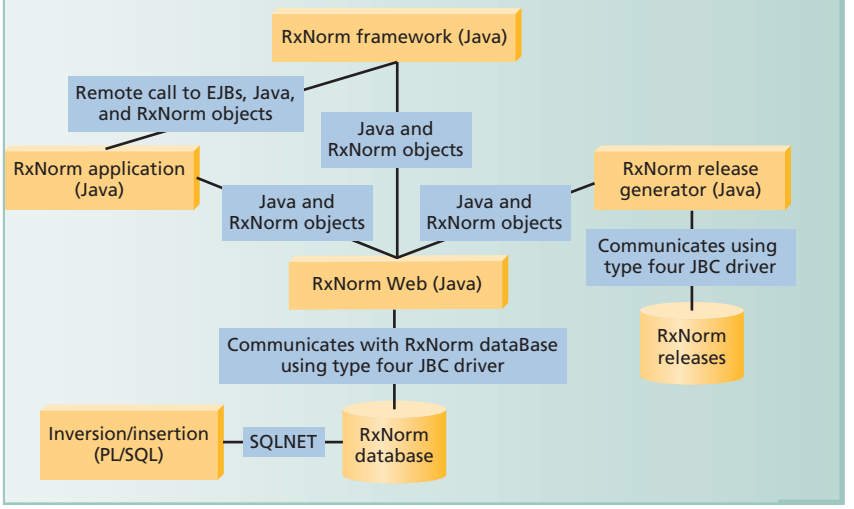
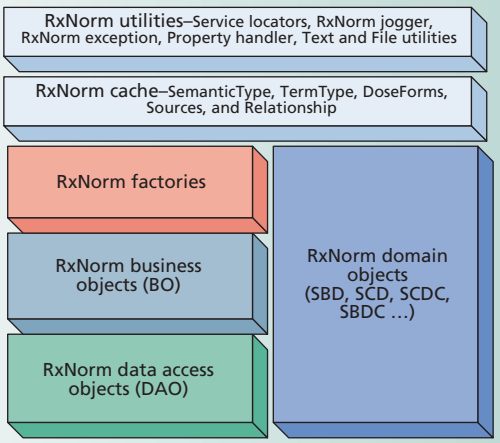


Table 1. Components used by the RxNorm subsystems.

	RxNorm subsystem		
Component	Inversion/ insertion	Editing	Production
RxNorm Framework (Java)		✓	✓
RxNorm Web (Java)		✓	
RxNorm App (Java)		✓	
Inversion/insertion (PL/SQL)	✓		
RxNorm ReleaseGenerator (Java)			✓

Figure 5. RxNorm framework.



consists of three major subsystems.

The *inversion/insertion subsystem* includes PL/SQL processes that invert data from various sources into common RRF format and then insert the processed data into the RxNorm database. (PL/SQL is a procedural language extension to SQL, Structured Query Language.) RxNorm editors then use the editing subsystem to edit these inserted data.

The *editing subsystem* is the core of the RxNorm system. It provides a Web interface that lets RxNorm editors create or edit semantic normal forms—SBD, SCD, SBDC, and the others. It has a three-tier implementation with an application server architecture, which allocates the main body of an application to run on a shared host rather than in the user system interface client environment. The application server does not drive the graphical user interfaces (GUIs); rather, it shares business logic, computations, and a data retrieval engine.

The *production subsystem,* implemented in a two-tier architecture, includes Java applications that generate the RxNorm release files in RRF. The two-tier architecture, which is frequently used for noncomplex, non–time-critical information processing systems, requires minimal operator intervention. The production applications produce three varieties of release files for download by licensed users: interval, cumulative, and full content.

RxNorm Software Component Architecture

Figure 4 shows the various RxNorm software components, their respective technologies, and how they interact with each other through a Type Four Java Database Connectivity (JDBC) driver. The Type Four JDBC driver converts JDBC technology calls into the network protocol used by database management systems (see the "For Further Information" sidebar for more on JDBC). This allows a direct call from the client machine to the DBMS server, making it a practical solution for intranet access. Table 1 shows the components that each RxNorm subsystem uses.

RxNorm Framework

The RxNorm framework is the support structure around which the NLM team organized and developed the other

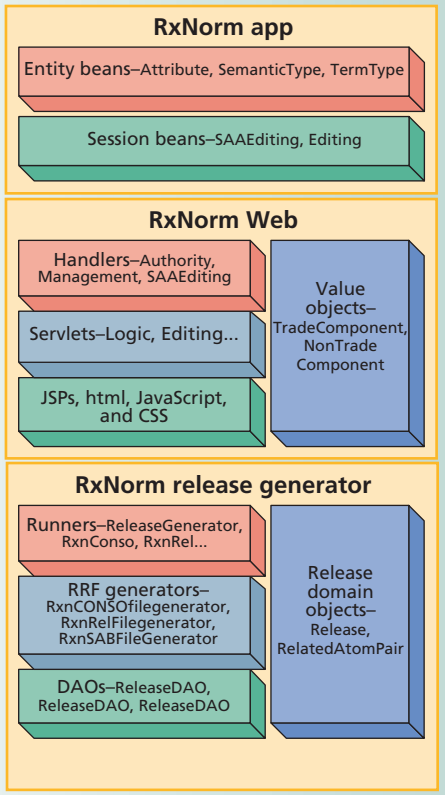
RxNorm Java components, and it encapsulates the core RxNorm business rules. The framework consists of RxNorm classes that are closely related in terms of function and data and that form an independent, reusable product. The RxNorm framework is a layered architecture: All other RxNorm Java components depend on the application logic layer (containing business objects), which in turn depends on a persistence layer (containing data access objects). Separating the code into these components and layers provides a level of encapsulation that makes the code more transparent and easier to maintain. Figure 5 shows the various layers within the RxNorm framework.

RxNorm *business objects* (BOs) represent tangible entities within an application that the user creates, accesses, and manipulates while performing a use case. BOs interact with each other to offer a range of functionalities. The term CRUD—"create, retrieve, update, and delete"—refers to the basic functions of a database or persistence layer in a software system. RxNorm BOs have clearly defined CRUD methods, which in turn use corresponding *data access objects* (DAOs). For example, an SBDBO has CRUD methods that use the SBDDAO's method for actual persistence operations such as creating, retrieving, updating, or deleting tangible entities. RxNorm BOs are all implemented as stateless, which means that their states do not persist between transactions.

RxNorm DAOs implement the access mechanism required to work with the RxNorm database. The BOs use

- **Java Database Connectivity (JDBC)—an API for the Java programming language that defines how a client may access a database. JDBC provides methods for querying and updating data in a database. See http://java.sun.com/j2se/1.5.0/docs/guide/jdbc/getstart/intro.html#1018466.**
- **Type Four JDBC driver—a native-protocol driver, fully enabled with Java technology, that converts JDBC technology calls into the network protocol directly used by database management systems (DBMSs). This allows a direct call from the client machine to the DBMS server and is a practical solution for intranet access. Since many of these protocols are proprietary, database vendors themselves will be the primary sources for this style of driver. See http://java.sun.com/products/jdbc/driverdesc.html.**
- **"A Semantic Normal Form for Clinical Drugs in the UMLS: Early Experiences with the VANDF," Stuart J. Nelson, Steven H. Brown, and colleagues; http://www.nlm.nih.gov/mesh/semanticnorm.html.**

Figure 6. RxNorm component overview.



the simpler interface exposed by these DAOs, and the DAO completely hides the data source implementation details from its clients. Because the interface exposed by the DAO to clients does not change when the underlying data source implementation changes, RxNorm DAOs can adapt to different storage schemes without affecting clients or business components. Essentially, a DAO acts as an adapter between the component and the data source.

RxNorm *domain objects* encapsulate persistent data. These are the real-world entities that the RxNorm framework manages—for example, SBD, SCD, TermType, SemanticType, and so on. The BOs create instances of this class, which the DAOs convert transparently into new rows in the database. The DAOs return instances of the appropriate domain class for each retrieved row of data.

The RxNorm *factories layer* constructs RxNorm domain or factory objects during runtime. The purpose of this design is to insulate the creation of objects from their usage. The factory object might decide the created object's class (if applicable) dynamically, return it from an object pool, do complex configurations on the object, or perform other operations. RxNorm factories contain the implementation of both the abstract factory pattern and the factory methods pattern.

RxNorm *caches* maintain the domain objects in a pool, providing a gateway for instant access to the system's persistent data, which is mostly static. RxNorm maintains caches of the following domain objects: TermTypes, SemanticTypes, DoseForms, relationships, sources, and attributes. Caches use the corresponding DAOs to retrieve these data from persistent store in the form of domain objects.

RxNorm *utilities* are classes that provide utility methods. The most frequently used utilities include,

- RxNormServiceLocator, which looks up remote home and local home interfaces for Enterprise Java Beans, data sources, and logging service from the underlying application container;
- StringUtil, which provides various string operations;
- RxNormLogger, used for logging purposes; and
- RxNormPropertyHolder, which loads properties defined in the property files into the runtime environment.

RxNorm Components

Each of these Java components built around the RxNorm framework has specific functionalities and follows the tiered (layered) architecture. Figure 6 shows the layers inside each of these three RxNorm components.

RxNorm Web is implemented in the Model-View-Controller (MVC) software architecture, which separates an application's data model, user interface, and control logic into three distinct components so that modifications to the view component can be made with minimal impact to the data model component. The views are the JSPs (Java servlet pages), HTML, and so on, and controllers are the servlets and handlers.

RxNorm App provides entity beans and stateless session beans. The *RxNorm Release Generator* generates the release file for RxNorm in RRF format.

THE RxNORM RECIPE

In creating RxNorm, our team followed best practices and design strategies. We adopted and implemented various design patterns proposed by Erich Gamma and colleagues (*Design Patterns,* Addison-Wesley, hardcover, 1995; CD, 1997). We also worked with J2EE patterns proposed by Deepak Alur, John Crupi, and Dan Malks (*Core J2EE Patterns: Best Practices and Design Strategies,* Prentice Hall/Sun Microsystems Press, 2001). Table 2 shows the RxNorm software components technology matrix.

FUTURE RESEARCH AND ENHANCEMENTS

We began with the goal of providing one RxNorm data release per month. This has helped us more fully understand the Java Technologies Web database and utility scripting intricacies of source inversion, and has given us the time to provide linking and files for download by our users. As our experience in performing these tasks grew,

Table 2. RxNorm software components technology matrix.

	Java technologies					Web			DB and utility scripting			
Component	Core Java	EJB	Servlets/ JSPs	JDBC	Junit	HTML	Css	Java script	SQL	PL/SQL	Ant	Unix scripts
RxNorm framework		✓			✓	✓				✓	✓	✓
RxNorm Web	✓		✓	✓	✓	✓	✓	✓	✓		✓	
RxNorm application	✓	✓		✓	✓				✓		✓	
RxNorm inversion/insertion	✓	✓		✓								
RxNorm production	✓			✓								✓

we identified data anomalies and inconsistencies, which we then worked with to create a usable model for consumer data use over time.

Future work will include expanding the RxNorm editing subsystem to give data editors streamlined methodologies for processing data in the system, providing a Web services interface to the RxNorm components, inserting new source vocabularies, and expanding SNF capabilities for electronic labels and other vocabularies. New data must be inserted and edited, pass through quality assurance, and be released to the public quickly and accurately. Thus, the editing subsystem must let the editors create and modify SNFs for new and existing data efficiently enough to yield data releases every week. Web services will give other medical systems (internal and external) the ability to access RxNorm data in real time.

Ultimately, our future research will concentrate on the data and how to make it clean, usable, highly available, and accurate. We will also focus on the practical use of the data produced; this, in turn, will yield insights into how to fine-tune our approach.

Although the electronic exchange of clinical information promises to be a multidepartmental, multidisciplinary endeavor, creating standards and tools to make clinically accurate, up-to-date data highly available is the right first step. Our work on the RxNorm project will help make this happen. We hope the RxNorm system will become a hub for medical—specifically pharmaceutical—vocabularies. ■

Simon Liu *is the director of Information Systems at the National Library of Medicine. Contact him at simon_liu@nlm.nih.gov.*

Wei Ma *is the chief of the Application Branch at the National Library of Medicine. Contact her at wei_ma@nlm.nih.gov.*

Robin Moore *is a project manager at the National Library of Medicine. Contact her at robin_moore@nlm.nih.gov.*

Vikraman Ganesan *is a software developer at the National Library of Medicine. Contact him at ganesav@mail.nih.gov.*

Stuart Nelson *is the head of the Medical Subject Headings Section at the National Library of Medicine. Contact him at stuart_nelson@nlm.nih.gov.*

For further information on this or any other computing topic, visit our Digital Library at http://www.computer.org/publications/dlib.