Building a Custom Rule Engine With Prolog

Rule-based and logic-based tools for encoding logical knowledge

Dennis Merritt

Databases and procedural programming languages are excellent for automating the data and procedures of an organization. However, they aren't very good for automating knowledge expressed as logical relationships or pattern-matching rules.

In this article, I describe the difficulties and benefits of switching from a procedural approach for encoding the rules for vaccination scheduling to a logic-base approach. I worked closely with Visual Data LLC developers and medical experts in the design and implementation of this application. I was responsible for the underlying Prolog code, the program interface to Delphi, and much of the initial knowledge engineering.

Dennis is the editor of the Dr. Dobb's AI Expert *newsletter and a principle of Amzi! Inc. He can be contacted at dennis@amzi.com.*

Office Practicum

Office Practicum is a software package from Visual Data LLC (http://www.visualdatallc.com/) that automates pediatric offices. It does all of the things you might expect, such as front-office services like automating billing and the calendar of appointments, and back-office services that make it easier for doctors to review and maintain a child's medical history. All of these tasks fall nicely into classic "data processing."

One particular Office Practicum data-gathering feature is an easy way of recording vaccinations children receive on visits, and associated reporting on the vaccination history. Doctors using Office Practicum liked the feature, but wanted it to go further. They asked Visual Data to add features that:

- Advise on the vaccines that could or should be given during a visit.
- Forecast future vaccinations for scheduling the next visit.
- Compare the vaccine history with regulatory information for reporting to schools and summer camps.

Knowledge about vaccination scheduling is contained in documents like the Centers for Disease Control's (CDC) periodical *Morbidity and Mortality Weekly Report* (*MMWR*), Department of Human Services (DHS) reports, drug company literature, and recommendations from the Advisory Committee on Immunization Practices (ACIP) and American Academy of Family Physicians (AAFP). This knowledge is usually expressed as a combination of:

- Definitions of vaccine names, types of vaccines, and commercial product names.
- Tables of schedules of doses and ages.
- Exception rules covering interactions with other vaccines.
- Rules for dealing with nonstandard vaccination schedules.

In other words, the knowledge is a collection of nonprocedural logical relationships—sometimes in rules, sometimes in tables, and sometimes as definitions.

Conventional Tools for Logical Knowledge

There is a tremendous temptation to use standard tools for encoding this type of knowledge, and that is exactly what Visual Data did at first. Visual Data is a Delphi shop, and its developers are comfortable with Delphi, and there's a lot to be said for sticking to a single development platform. Consequently, Visual Data pushed ahead with a pure Delphi solution to encoding the vaccination logic.

But it was difficult. The primary problem was that the flow-of-control branching *if-then* statements of a procedural language did not have the same semantics as the pattern-matching *if-then* rules of

logical relationships. Developers must make arbitrary decisions of where in the flow-of-control to code each branch, thereby entangling the declarative rules in the thread of execution.

In the case of the vaccine knowledge, this was made more difficult by the interactions that can happen between vaccines, and the fact that many vaccines are actually combination vaccines covering multiple diseases. For example, a child might get a measles vaccine and a rubella vaccine; or a combined measles, mumps, and rubella (MMR) vaccine. Because measles is a live virus vaccine, it can't be given within a certain time period after any other live virus vaccine has been given; and vice versa for other live virus vaccines.

Another difficulty was that the rules were expressed using date intervals and ages. Intervals and ages could be expressed in days, weeks, months, or years. To simplify the procedural coding, they converted all intervals to days, but this led to problems as months aren't all the same length. Doctors complained that if they tried on March 15 to give a two-month dose to a child born on January 15, the software would say the child wasn't two months old yet.

Visual Data did, however, get it to work, although it didn't do everything they wanted—it was difficult to write, and the 5000+ lines of Delphi code were abstruse at best. Customers loved it, but were bothered by the date problems and wanted to know when the software would support the new combination vaccine, Pediarix, which included DTaP, hepatitis B, and polio vaccines (DTaP is itself a combination of diphtheria, tetanus, and pertussis vaccines).

Specialized Tools for Logical Knowledge

At this point, Visual Data decided to look at options for encoding the vaccination logic. Clearly, some sort of declarative, rule-based tool would let them code the knowledge in a manner that was easier to maintain.

Developers looked at a number of tools available for rule-based programming, but they were all either designed for a specific application context, or were general-purpose tools not particularly well suited to encoding the vaccination knowledge. Many were also prohibitively expensive. In the end, they opted to use the logic programming language, Prolog, because it allowed for a much more flexible solution specifically adapted to their needs, and it could be easily integrated in the Delphi application.

Two Faces of Prolog

There are two ways to use Prolog for automating logical knowledge. One is to code the knowledge directly in Prolog rules and rely on Prolog's built-in reasoning strategy; the other is to use Prolog to create knowledge structures and reasoning strategies custom tailored for the application.

The advantages of the first approach are that Prolog rules are easy to learn and apply, and compiled Prolog execution is extremely fast. The main disadvantage is that the rules have precise semantics, which may or may not be well-suited for a particular application.

The advantages of the second approach are that the knowledge representation and reasoning strategy can be perfectly tuned for the application and can be developed quickly due to Prolog's representational power for building such applications. The disadvantage is that it is more difficult to learn to use Prolog for crafting custom knowledge base tools, and the runtime execution of the rules will be in an interpretive rather than compiled mode.

Architecture

As Figure 1 illustrates, Visual Data took a hybrid approach when building its "VacLogic" engine. The definitions and tables were customized Prolog structures with underlying Prolog code to support their use. The rules, on the other hand, used Prolog's native syntax, but were made easier to write using a number of utility predicates that made it easy to reason over date and age quantities.

A Prolog file called the "reasoner" served as the engine for the application. It was written by the software developers and is

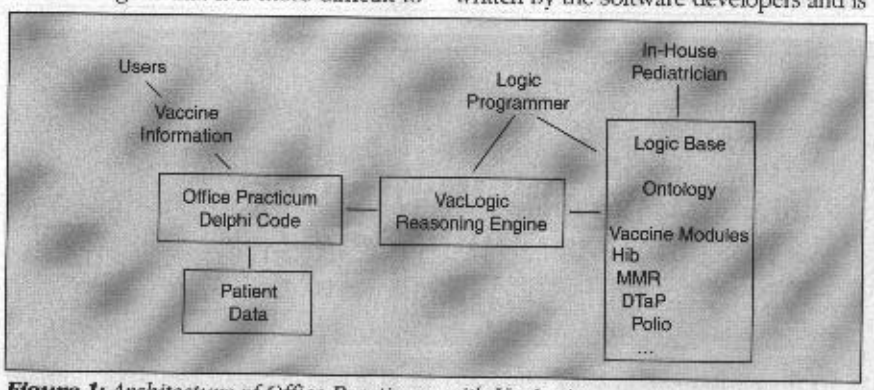
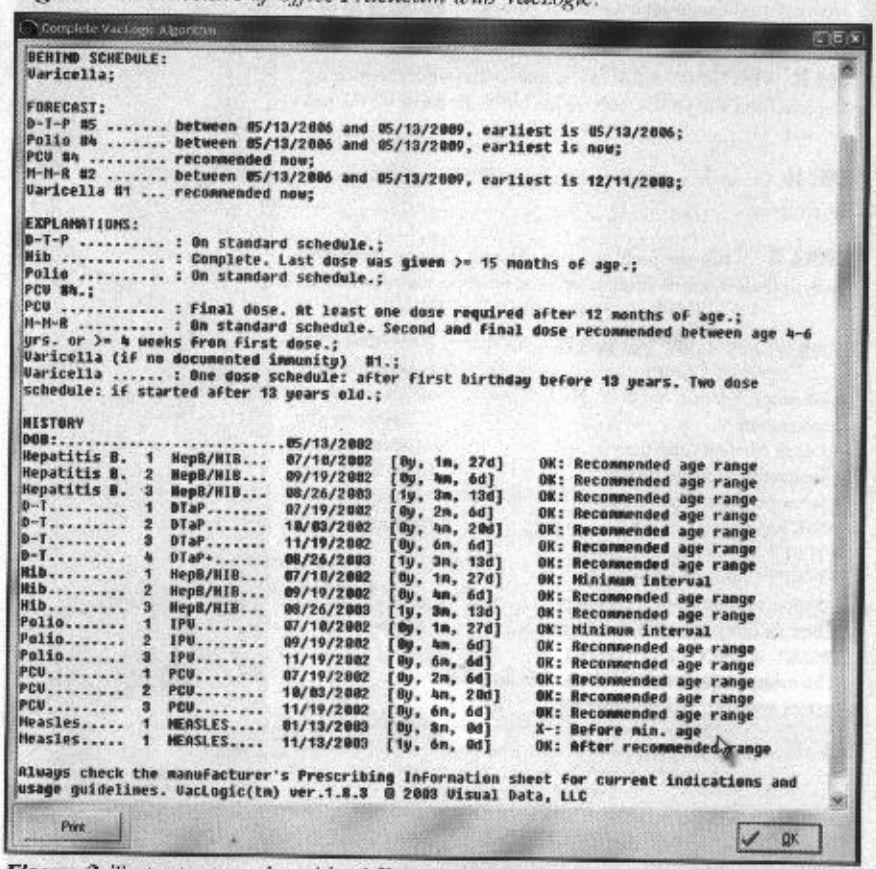


Figure 1: *Architecture of Office Practicum with VacLogic.*



```
BEHIND SCHEDULE:
Varicella;

FORECAST:
D-T-P #5 ..... between 05/13/2006 and 05/13/2009, earliest is 05/13/2006;
Polio #4 ..... between 05/13/2006 and 05/13/2009, earliest is now;
PCV #4 ..... recommended now;
M-M-R #2 ..... between 05/13/2006 and 05/13/2009, earliest is 12/11/2008;
Varicella #1 ... recommended now;

EXPLANATIONS:
D-T-P ..... : On standard schedule.;
Hib ..... : Complete. Last dose was given >= 15 months of age.;
Polio ..... : On standard schedule.;
PCV #4.;
PCV ..... : Final dose. At least one dose required after 12 months of age.;
M-M-R ..... : On standard schedule. Second and final dose recommended between age 4-6
yrs. or >= 4 weeks from first dose.;
Varicella (if no documented immunity) #1.;
Varicella ..... : One dose schedule: after first birthday before 13 years. Two dose
schedule: if started after 13 years old.;

HISTORY
DOB:.....05/13/2002
Hepatitis B.  1  HepB/HIB...  07/18/2002  [0y, 1m, 27d]   OK: Recommended age range
Hepatitis B.  2  HepB/HIB...  09/19/2002  [0y, 4m, 6d]   OK: Recommended age range
Hepatitis B.  3  HepB/HIB...  08/26/2003  [1y, 3m, 13d]  OK: Recommended age range
D-T.....      1  DTaP.....    07/19/2002  [0y, 2m, 6d]   OK: Recommended age range
D-T.....      2  DTaP.....    10/03/2002  [0y, 4m, 20d]  OK: Recommended age range
D-T.....      3  DTaP.....    11/19/2002  [0y, 6m, 6d]   OK: Recommended age range
D-T.....      4  DTaP+....    08/26/2003  [1y, 3m, 13d]  OK: Recommended age range
Hib.....      1  HepB/HIB...  07/18/2002  [0y, 1m, 27d]  OK: Minimum interval
Hib.....      2  HepB/HIB...  09/19/2002  [0y, 4m, 6d]   OK: Recommended age range
Hib.....      3  HepB/HIB...  08/26/2003  [1y, 3m, 13d]  OK: Recommended age range
Polio.....    1  IPV.....     07/18/2002  [0y, 1m, 27d]  OK: Minimum interval
Polio.....    2  IPV.....     09/19/2002  [0y, 4m, 6d]   OK: Recommended age range
Polio.....    3  IPV.....     11/19/2002  [0y, 6m, 6d]   OK: Recommended age range
PCV.....      1  PCV.....     07/19/2002  [0y, 2m, 6d]   OK: Recommended age range
PCV.....      2  PCV.....     10/03/2002  [0y, 4m, 20d]  OK: Recommended age range
PCV.....      3  PCV.....     11/19/2002  [0y, 6m, 6d]   OK: Recommended age range
Measles.....  1  MEASLES....  01/13/2003  [0y, 8m, 0d]   X-: Before min. age
Measles.....  1  MEASLES....  11/13/2003  [1y, 6m, 0d]   OK: After recommended range

Always check the manufacturer's Prescribing Information sheet for current indications and
usage guidelines. VacLogic(tm) ver.1.8.3  @ 2003 Visual Data, LLC
```

Figure 2: *Text report produced by Office Practicum for camp or school use.*

not intended to be maintained by the application domain experts. It contains:

- The application program interface predicates, called from Delphi for vaccine services.
- The implementation of various utility predicates that handle date and age expressions.
- The reasoning support for the definitional and tabular knowledge structures.
- The predicates that are used to derive higher-level concepts from lower-level ones.
- The basic strategy for using the rules to get the status of the various vaccinations.

A Prolog file called "general" contained the definitions of various vaccines, as well as citation information used for vetting the knowledge. It is intended for use by in-house pediatric experts.

Each individual vaccine has a module that contains the tables and rules that apply to that vaccine. When a module is consulted, the resulting output is stored with all the pertinent data about the vaccine, such as the next dose, the minimum date for the next dose, whether it is due today or up-to-date.

General Knowledge

The general knowledge is stored in an "ontology." Ontologies provide a way to store definitions of related concepts so that those concepts can be readily used by an application. In this case, the ontology provides a way to record hierarchies of vaccine types and other types of relationships, such as critical definitions of which vaccines contain live viruses.

Listing One presents some sample entries indicating that Varicella and Small Pox are live viruses: the three elements of the new Pediarix vaccine, various spellings of haemophilus influenzae (Hib), and some of the PRP-OMP flavors of Hib. These are in an *is-a* hierarchy of terms, so a rule referring to Hib correctly matches a data record referring to the vaccine PedvaxHIB, because it *is-a* PRP-OMP, and PRP-OMP *is-a* Hib.

The idea behind this representation is that it becomes easy to add new entries for different spellings, new vaccines, and new categories of vaccine that are needed by the rules. In-house pediatric experts can directly examine the definitions and make additions and modifications as necessary.

This is clearly an advantage over the procedural approach, and also a big advantage over rule-based approaches that do not provide ontologies. In a pure rule-based approach, these definitions would all have to be implemented as rules. Having the writers of the rules stick to a limited vocabulary doesn't work because so many different vaccines and spellings appear in the medical records. It is the ontology that makes it practical to use the existing medical records.

Tables

The vaccine logic base is used to report on vaccines that can be given at a visit, and to forecast vaccines needed in the future. There is both a minimum and recommended age for the forecast vaccines, as well as a minimum separation from the last vaccination. These are reflected in the table entries.

Listing Two is a table in the Hib module for Hib vaccines that contain PRP-OMP. There are other tables for Hib reflecting other flavors of the Hib vaccine and schedules that weren't started on time. As with the ontology entries, tables are intended to be easily reviewed and modified by in-house pediatric experts. They also provide similar advantages over a pure rule-based approach. This same knowledge could be encoded as multiple *if-then* rules, but it would not be as easy to verify or maintain.

Further, because the tables can use any date units, they can be entered almost exactly as they appear in the original documents, without need for conversions.

Rules

The rules use patterns in the data to determine the status of a particular vaccination, deciding whether or not to use a table, and how to qualify the results from the

table, if necessary. It would have been possible to generate a rule syntax customized for this application, but because the rules are crisp goal-driven rules that map nicely to Prolog native rules, Prolog syntax was used. This approach also had the advantages of faster execution time and shorter development time, but the cost is that the rules are not as clear as they could have been with a custom rule syntax.

The rules are, however, compact and closely reflect the actual knowledge because of the helper predicates in the reasoner that support dates and date intervals.

Each module has as its main predicate *status/0*, which is called by the reasoning engine when it is time to get the information about a particular vaccine. The *status/0* predicate might defer to other predicates to get the final results for a vaccine.

Listing Three is a status rule in the Hib module that defers to *late_start_status* rules if the child has had two valid Hib vaccines but the second one was given after seven months of age. (Other rules will have previously examined the past history and determined which vaccines were given at valid times.)

A rule that generates output usually has three parts: the conditions under which the rule applies, the calculations for status and dates, and the storing of key information about the vaccine. Listing Four presents the conditions, calculations, and output expanded for a sample MMR rule.

The conditions rule in Listing Four(a) applies if there have been measles vaccinations and there have been more than one live virus vaccination given.

The next dose is one more than the current count and the table from *MMWR* published by the Centers for Disease Control (CDC) is used to get the minimum age and recommended age for the next vaccination. The minimum spacing is not retrieved from the table because the rules for live virus spacing override it. The calculations rule in Listing Four(b) easily refers to the last live virus vaccination. This is made possible by the ontology. Also, the recommended age is really a range, with a lower and upper bound. Once the recommended dates are determined, a utility predicate in the reasoning engine is called to determine the status for a visit today. This might indicate it's up to date, due, or not due yet but okay to give today.

As with the tables and ontology, the idea is to make the rules as readable as possible, but they do start to get complex. On the other hand, the complexity is exactly equal to the complexity of the rule itself.

While the goal of the system was to allow in-house experts to work these rules as well, they did get a little too programmatic. However, they can still be read and critiqued by experts during the vetting process.

The output portion of the rule, see Listing Four(c) and Figure 2, directs the reasoning engine to store certain information to be returned to the calling program in response to various requests. The note is used in verbose reporting and is written by the in-house expert. The dates, dose, and status are used for various reports.

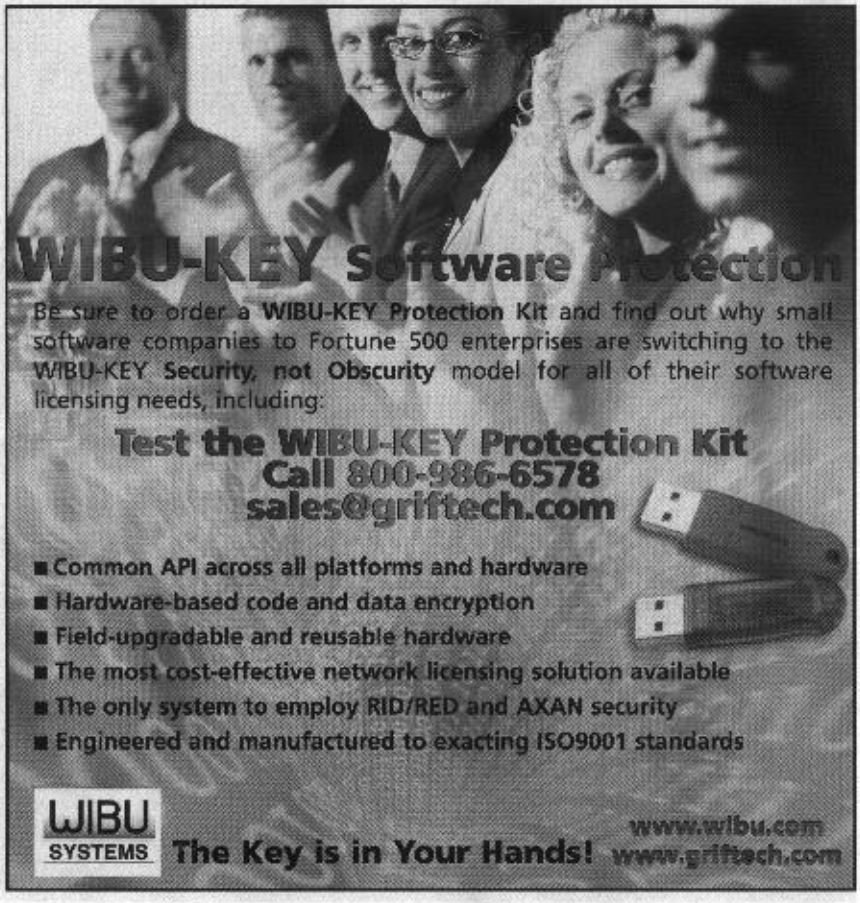
Testing Tools

The reasoning engine and logic base modules have a test harness that is used for testing and debugging in a Prolog environment. This allows the use of Prolog trace and debugging tools, as well as other development tools such as predicate cross references and outliners.

Test cases are stored in a file using ages for inputs that are used by the test harness as offsets for a randomly generated starting date.

Listing Five is test data that is used to drive the MMR and Varicella modules. The child is 18 months old and has had two MMR vaccinations. The random dates used for this run (Listing Six) have the child born on 5/13/2002 and the date of the office visit as 11/13/2003. The output from the test indicates the types of data available from the vaccine logic base, including the history analysis feature of the system.

In the HISTORY report, the first measles vaccination was given too early and does



not count, so for planning purposes, the MMR dose #2 is next and the appropriate dates for that dose are provided. Varicella wasn't given when it should have been, and is behind schedule and due today. The testing interface also provides for regression testing, so previous test cases can be automatically run and compared against known results.

Delphi Interface

For a given child, the Delphi program first goes to the database and extracts the pertinent information about past vaccinations. That data is then asserted to the logic base.

The logic base is then queried from different parts of the Delphi code for different reasons. For example, Figure 3 shows one part of the code displays for the doctor what vaccinations are due and/or allowed on today's visit. Another displays future requirements for scheduling follow-up office visits. Another is used to generate camp, school, and other regulatory forms needed for reporting on a child's vaccination history.

Future Plans

The development environment provides tools for testing and debugging the logic base, but uses text files for the various modules. The structure of the various logical knowledge entities would map nicely to a graphical development environment, and this is planned as a future project.

Now that the logic base is a separate unit, it can be deployed and used in different contexts. Visual Data is considering offering the history analysis portions of the vaccine logic base on the Web using a .NET architecture.

Cost Benefit

The benefits from the logic-base approach include:

- A 90 percent reduction in code used for vaccine logic rules.
- Direct access to the knowledge by in-house pediatricians.
- Localization of all the vaccine logic, which used to be scattered in the different parts of the application with different needs.
- Easy maintenance and quality-assurance testing.
- Additional capabilities that were too hard to encode before, such as the complete analysis of past vaccination history and support for new multivaccine products.

All of these benefits add up to the one major benefit, which is that Visual Data's software now provides better services for customers in this area, which is critically

important in the running of a pediatric office.

The costs were:

- Time spent investigating and learning about various alternative approaches for encoding the vaccination logic.
- Software license fees.
- A few months of development time.
- Time spent learning the new technology.

The calendar time for the project was:

- January: Start study of alternative approaches.
- March: Begin application development.
- June: Deploy to the field.

Conclusion

Unlike factual or procedural knowledge, logical knowledge is difficult to encode in a computer. However, the ability for organizations to successfully encode their logical knowledge can lead to better services for users. The question, then, is how best to encode logical knowledge. It can be shoe-horned into data- and procedure-based tools, but the encoding is difficult, the knowledge becomes opaque, and maintenance becomes a nightmare. Rule-based and logic-based tools are better suited to the encoding of logical knowledge, but require the selection of the proper tool for the knowledge to be encoded, and the learning of how to use that tool. Off-the-shelf rule or logic tools sometimes provide a good solution, but often the knowledge representation of the tool doesn't fit the actual knowledge, or the reasoning engine doesn't use the knowledge as it is supposed to be used. This leads to the same coding and maintenance problems experienced with conventional tools, but to a lesser extent depending on how big the semantic gap is between the knowledge and the tool.

A viable alternative is the building of a custom logic-based language and reasoning engine. This allows for the closest fit between the coding of the knowledge and the actual knowledge, and for the cleanest integration between the tool and the rest of the application context.

DDJ

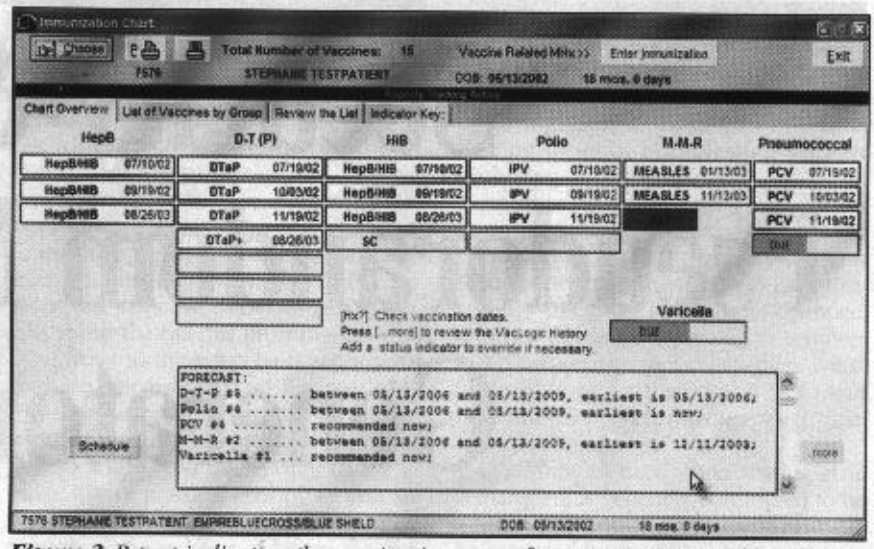


Figure 3: *Report indicating the vaccination status for a test patient; used by the doctor on the day of visit.*

Listing One

```
live_virus ->> 'Varicella'.
live_virus ->> 'Small Pox'.

'DTaP-HepB-IPV+' ->> 'Pediarix(tm)'.

'HIb' ->> 'HbOC'.
'HIb' ->> 'PRP-OMP'.
'HIb' ->> 'PRP-T'.

'PRP-OMP' ->> 'PRP-OMP & Hep B'.
'PRP-OMP' ->> 'ProvaxHI3(tm)'.
'PRP-OMP' ->> 'HIB-OMP'.
```

Listing Two

```
table('Recommended 8', [
% Recommended Schedule B from 'DOS Hib 2003Mar' for vaccines
% containing PRP-OMP
%   Dose      Minimum      Minimum      Recommended
%               Age         Spacing        Age
    [  1.      6 weeks,     none,        2 months ],
    [  2.     10 weeks,     4 weeks,     4 months ],
    [  3.     12 months,    8 weeks,    15 months ]  ]).
```

Listing Three

```
status :-
    valid_count('Hib') gte 2,
    vaccination(2, 'Hib') #>= 7 months,
    !,
    late_start_status.
```

Listing Four

(a)

```
valid_count('Measles') eq Count,
valid_count(live_virus) gt 0,
```

(b)

```
    NextDose is Count + 1,
    get_row('M-M-R':'MMWR Recommended', [NextDose, MinAge, _, RecAge, _]),
    MinDate #= maximum( vaccination(last,live_virus) + 4 weeks,
                        birthday + MinAge).
    RecDate1 #= maximum( vaccination(last,live_virus) + 4 weeks,
                         birthday + lower(RecAge)),
    RecDate2 #= maximum( vaccination(last,live_virus) + 4 weeks,
                         birthday + upper(RecAge)),
    calc_status(MinDate, RecDate1, RecDate2, Status),
```

(c)

```
    output('M-M-R', [
        dose = NextDose,
        dates = [MinDate, RecDate1, RecDate2],
        rec_info = RecAge,
        status = Status,
        citation = 'MMWR 2002Feb08',
        note = [
          `Next dose based on standard schedule with `,
          `guaranteed 4 week separation from last `,
          `live virus vaccination.` ]]).
```

Listing Five

```
test(mv06, vaccines, ['M-M-R', 'Varicella']).
test(mv06, comment, ['First MMR too early to valid_count,
                                    ready for real #2; Varicella overdue']).
test(mv06, age, 10 months).
test(mv06, 'M-M-R', vaccination, 8 months).
test(mv06, 'M-M-R', vaccination, 18 months).
```

Listing Six

```
STATUS
  Varicella (if no documented immunity)  #1.
PLANNING
  M-M-R #2 between 05/13/2006 and 05/13/2009, earliest is 12/11/2003
  Varicella #1 recommended now
BEHIND SCHEDULE
  Varicella
HISTORY
  Measles:1:M-M-R:01/13/2003;
      [8 years, 8 months, 0 days]:X-: Before min. age
  Measles:1:M-M-R:11/13/2005;
      [1 years, 6 months, 0 days]:OK: After recommended range
```

DDJ