CSC 868.01 – Spring 2013
Advanced Object Oriented Software Design and Development
Prof. Levine

DSS1 Interpreter

(4/28 Milestone)

Group 1

April 28, 2013

Group members

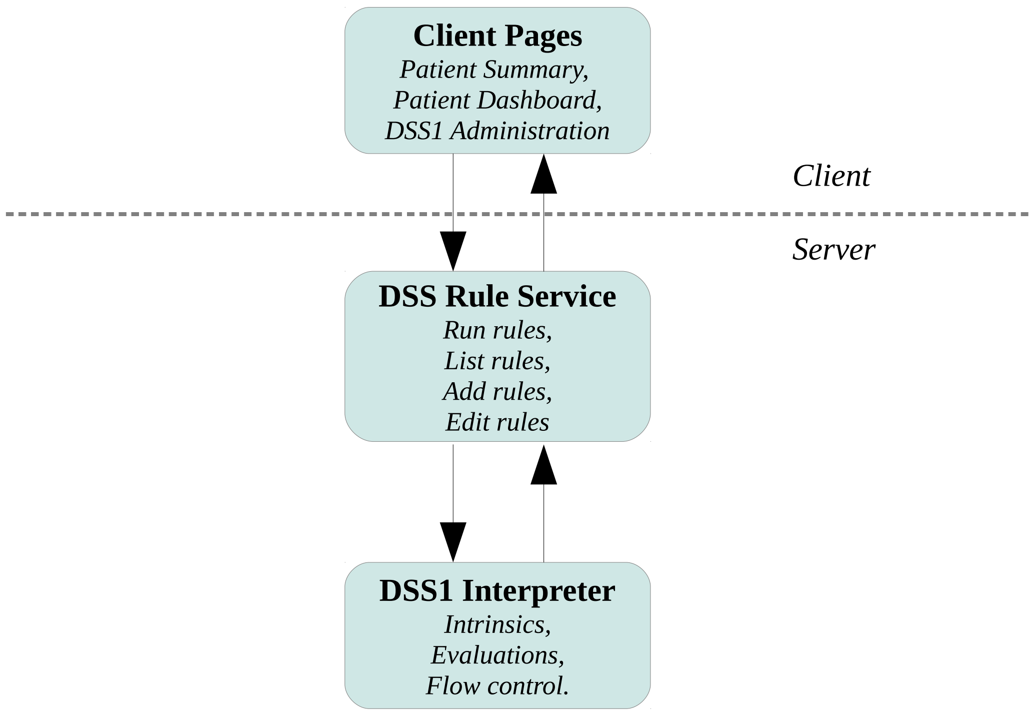
Robert (Bob) Bierman
Victor Woeltjen
Jason Lum
Steven Gimeno
Ying Kit Ng (Kent)
Bianca Uy
Kay Choi

Overview

This document describes the implementation of DSS1 intrinsic functions and the integration of rules into relevant web pages in the OpenMRS system.

Architecture

The DSS1 rule subsystem is incorporated into OpenMRS in a simple Client-Server fashion. The target implementation will feature client-side web pages which interact with the DSS1 rule subsystem on the server by way of DSSRuleService.



While the DSS Rule Service runs on the server, its interface is exposed to client-side JavaScript code via DWR (Direct Web Remoting).

Note that this is an intermediary milestone in a transition from a Model-View-Controller architecture to the simplified Client-Server approach. Certain classes used in this milestone (such as Controller classes to handle uploading and saving rules) will be made obsolete by future pages which may interact with the DSS Rule Service directly.

Client Pages

The following web pages interact with the DSS Rule Service:

- **Patient Summary** *(patientsummary.jsp)***:** A stand-alone page, reachable from a link on the Patient Dashboard. Used primarily to contain major information about a patient (gender, age, WHO stage, etc.) Invokes the rule service via DWR to retrieve all alerts for the named target “summary” and displays them below other patient information.
- **Patient Dashboard** *(org.openmrs.module.basicmodule.extension.html.PatientSummaryExtension):* Primary landing point for viewing patient information; contains multiple tabs for this purpose. This extension inserts a link to the Patient Summary on the Patient Dashboard, and accompanies this with relevant alerts by invoking the rule service for the “dashboard” target.
- **Create DSS Rule** *(dssRules.form)**:*** Provides a form where DSS source code can be entered and submitted to the rule service with a specific rule name. Future enhancements will consolidate the ability to create new rules, load existing rules, and edit rules in one form. Made accessible through an extension to the Administration menu.
(Note: Currently this interacts via the class DSSRuleController; future versions may be simplified to interact with the rule service directly via DWR.)

DSS Rule Service

The DSSRuleService follows the facade design pattern to expose the following functionality to clients. The high-level tasks that are relevant to client code are defined using a few simple methods which hide the details of compiling, interpreting, and managing the storage of rules.

Method	*Description*	*Details*
store(rule, code)	Stores a rule (either as a new rule, or replacing an existing rule) with the given source code.	Invokes the Parser to convert source code to AST; Invokes the XMLBuilder to convert AST to DOM and save; Saves the original source to file system for subsequent retrieval; Stores the AST in memory for subsequent running.
load(rule)	Load the source code for an existing rule.	Reads stored source code from the file system.
listRules()	List all existing rules.	Returns a list of all stored rule names.
runRules(patientId, target)	Get all alerts for the given target (“summary” or “dashboard”) as appropriate to the given patient.	For each rule: Construct interpreter; Install intrinsics, including “alert” function which stores to a map; Pre-define “patientId” for DSS1 program; Run the interpreter on the rule. Thereafter, pull all alerts appropriate to the target from the map.

Interpreter.

The interpreter is implemented using the Visitor design pattern, traversing the Abstract Syntax Tree (AST) produced by the existing Compiler using an implementation of the provided ASTVisitor interface, performing computation as appropriate at every given node in the tree.

The Visitor design pattern leverages double dispatch to decouple a data structure from the operations which can be performed while traversing this data structure. The Visitor calls an “accept” method on a node within the data structure, which is itself overloaded to call a more specific method on the Visitor itself; “visitBlockTree”, for example. This permits the external object – the Visitor – to implement behavior using the data structure's type hierarchy, without adding that specific behavior to those types directly.

In the case of the Interpreter, the data structure is the AST, which describes a DSS1 program as a tree of elements – block (BlockTree), if statements (IfTree), et cetera. The Visitor is the IntepreterVisitor, which manages and performs the computation described by this program. This is done with the support of other underlying subsystems to describe variable state and perform type-specific evaluations, as described in the Architecture section.

Conventions

In some cases, the meaning of specific types of nodes is context-dependent. For instance, an IdTree may occur either as an expression, or as the left part of an assignment, or in function arguments. For consistency, the visitation behavior for this and similar nodes is implemented as the evaluation of an expression or similar – that is, the most computational, least structural interpretation as a node. As such, visitIdTree shall return the value currently stored in the variable described by the identifier; other uses of IdTree must be handled by their parents in the tree. For instance, the AssignTree inspects its left-hand child and, when this is an IdTree, performs storage to the variable it describes.

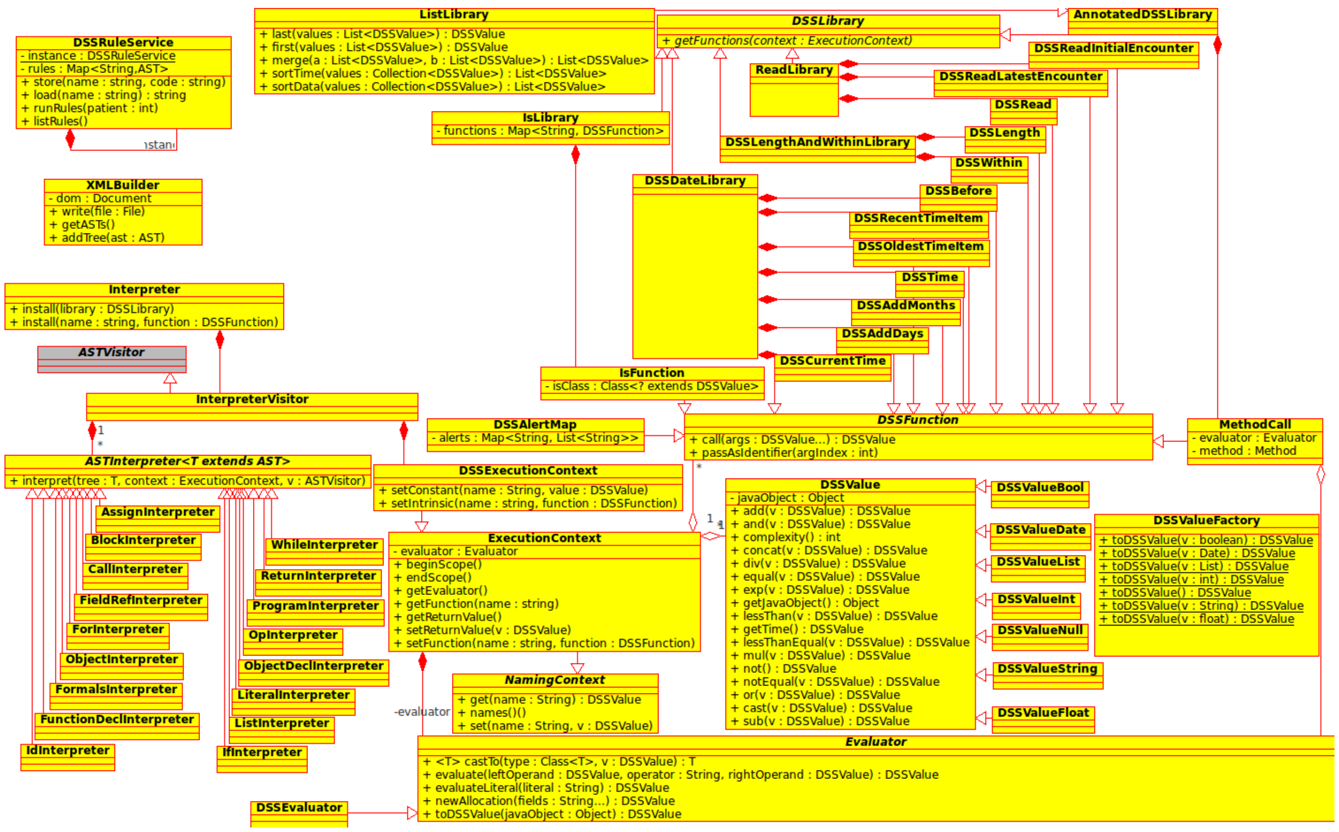
Interpreter subsystems

The interpreter is implemented as four major subsystems:

- **Traversal subsystem:** InterpreterVisitor and other used classes. This subsystem is responsible for high-level interpretation of the program, including flow control, and coordinating complex expressions.
- **Execution context:** ExecutionContext. Used to store state relevant to the execution. This includes the current values of variables, and all defined functions (intrinsic or otherwise).
- **Evaluation subsystem:** Evaluator, its implementation, and several objects to represent types within DSS1. This subsystem is responsible for implementing the semantics of different types and of specific operations upon such values.
- **Intrinsic libraries:** DSSLibrary and its implementations. These expose Java implementations of intrinsics to DSS1 programs.

The current implementation effectively uses these subsystems in tiers. An Evaluator is provided to the ExecutionContext upon construction; this ExecutionContext is provided to the InterpreterVisitor on *its* construction. Note that these subsystems therefore require no knowledge of the systems “above” them (the execution context does not know about traversal; the evaluation subsystem is entirely self-contained).

Class Diagram.


DSSRuleService
- instance : DSSRuleService
- rules : Map<String,AST>
+ store(name : string, code : string)
+ load(name : string) : string
+ runRules(patient : int)
+ listRules()
XMLBuilder
- dom : Document
+ write(file : File)
+ getASTs()
+ addTree(ast : AST)
Interpreter
+ install(library : DSSLibrary)
+ install(name : string, function : DSSFunction)
ASTVisitor
InterpreterVisitor
ASTInterpreter<T extends AST>
+ interpret(tree : T, context : ExecutionContext, v : ASTVisitor)
AssignInterpreter
BlockInterpreter
CallInterpreter
FieldRefInterpreter
ForInterpreter
ObjectInterpreter
FormalsInterpreter
FunctionDeclInterpreter
IdInterpreter
WhileInterpreter
ReturnInterpreter
ProgramInterpreter
OpInterpreter
ObjectDeclInterpreter
LiteralInterpreter
ListInterpreter
IfInterpreter
DSSEvaluator
ListLibrary
+ last(values : List<DSSValue>) : DSSValue
+ first(values : List<DSSValue>) : DSSValue
+ merge(a : List<DSSValue>, b : List<DSSValue>) : List<DSSValue>
+ sortTime(values : Collection<DSSValue>) : List<DSSValue>
+ sortData(values : Collection<DSSValue>) : List<DSSValue>
IsLibrary
- functions : Map<String, DSSFunction>
DSSLibrary
+ getFunctions(context : ExecutionContext)
AnnotatedDSSLibrary
DSSReadInitialEncounter
ReadLibrary
DSSReadLatestEncounter
DSSRead
DSSLengthAndWithinLibrary
DSSLength
DSSWithin
DSSDateLibrary
DSSBefore
DSSRecentTimeItem
DSSOldestTimeItem
DSSTime
DSSAddMonths
DSSAddDays
DSSCurrentTime
IsFunction
- isClass : Class<? extends DSSValue>
DSSAlertMap
- alerts : Map<String, List<String>>
DSSFunction
+ call(args : DSSValue...) : DSSValue
+ passAsIdentifier(argIndex : int)
MethodCall
- evaluator : Evaluator
- method : Method
DSSExecutionContext
+ setConstant(name : String, value : DSSValue)
+ setIntrinsic(name : string, function : DSSFunction)
ExecutionContext
- evaluator : Evaluator
+ beginScope()
+ endScope()
+ getEvaluator()
+ getFunction(name : string)
+ getReturnValue()
+ setReturnValue(v : DSSValue)
+ setFunction(name : string, function : DSSFunction)
NamingContext
+ get(name : String) : DSSValue
+ names()()
+ set(name : String, v : DSSValue)
-evaluator
DSSValue
- javaObject : Object
+ add(v : DSSValue) : DSSValue
+ and(v : DSSValue) : DSSValue
+ complexity() : int
+ concat(v : DSSValue) : DSSValue
+ div(v : DSSValue) : DSSValue
+ equal(v : DSSValue) : DSSValue
+ exp(v : DSSValue) : DSSValue
+ getJavaObject() : Object
+ lessThan(v : DSSValue) : DSSValue
+ getTime() : DSSValue
+ lessThanEqual(v : DSSValue) : DSSValue
+ mul(v : DSSValue) : DSSValue
+ not() : DSSValue
+ notEqual(v : DSSValue) : DSSValue
+ or(v : DSSValue) : DSSValue
+ cast(v : DSSValue) : DSSValue
+ sub(v : DSSValue) : DSSValue
DSSValueBool
DSSValueDate
DSSValueList
DSSValueInt
DSSValueNull
DSSValueString
DSSValueFloat
DSSValueFactory
+ toDSSValue(v : boolean) : DSSValue
+ toDSSValue(v : Date) : DSSValue
+ toDSSValue(v : List) : DSSValue
+ toDSSValue(v : int) : DSSValue
+ toDSSValue() : DSSValue
+ toDSSValue(v : String) : DSSValue
+ toDSSValue(v : float) : DSSValue
Evaluator
+ <T> castTo(type : Class<T>, v : DSSValue) : T
+ evaluate(leftOperand : DSSValue, operator : String, rightOperand : DSSValue) : DSSValue
+ evaluateLiteral(literal : String) : DSSValue
+ newAllocation(fields : String...) : DSSValue
+ toDSSValue(javaObject : Object) : DSSValue
1
*
1
*

Traversal subsystem

- **InterpreterVisitor:** Provides an implementation of the ASTVisitor interface. Maintains several ASTInterpreter objects (one for each type), to which calls to the various visit methods are delegated. (This arrangement is primarily for purposes of code organization.)
  - ***ASTInterpreter:*** Interface used when interpreting specific node types. An ASTInterpreter knows how to interpret some specific type of node, as determined by concrete implementations. Note that, in addition to the relevant node, an ExecutionContext and an ASTVisitor are passed as arguments when this behavior is invoked. This allows implementations to consult variable state, retrieve the Evaluator, traverse the tree, et cetera. Examples follow:
    - **AssignInterpreter:** Handles variable assignment.
    - **BlockInterpreter:** Runs a block of code. Polls for return values in the ExecutionContext to exit early after a *return* has been encountered. (This state will eventually be cleared when control reaches the end of the user-defined function.)
    - **CallInterpreter:** Retrieves a named function from the ExecutionContext and invokes it, returning its return value.
    - …
    - **ReturnInterpreter:** Stores a return value to the ExecutionContext.
    - **WhileInterpreter:** Evaluates an expression, and continues running until that expression is false.

The implementation of WhileInterpreter is presented here as an example:

```
public class WhileInterpreter implements ASTInterpreter<WhileTree> {
    public Object interpret(WhileTree tree, ExecutionContext context, ASTVisitor visitor) {
        // Evaluate the condition; use the result as a Java while's condition
        while (evaluate(tree.getKid(1), context, visitor)) {
            tree.getKid(2).accept(visitor); // Traverse to block child
        }
        return null;
    }

    // Helper function to evaluate the condition tree to return a Java boolean
    private boolean evaluate(AST t, ExecutionContext context, ASTVisitor v) {
        Object result = t.accept(v);
        if (result instanceof DSSValue) {
            // Convert the result to a Java boolean
            return context.getEvaluator().castTo(Boolean.class, (DSSValue)result);
        }
        // Default to false
        return false;
    }
}
```

ExecutionContext

- **ExecutionContext:** Provides a means to store and retrieve: Return values; variable states; and named functions. Handles rules of scope to hide variables during function calls. Also exposes the Evaluator. Note that this may be populated with functions or even variables before being given to the InterpreterVisitor, allowing the definition of intrinsics.
    - ***DSSFunction:*** An interface used to describe any function called from DSS1 (either intrinsic or user-defined). This permits function calling to be implemented identically for both categories of function. Additionally includes a method for testing if a given argument should be passed as a raw identifier instead of evaluated directly (as used by some intrinsics.).
        - *User-defined functions:* Defined by an inner class of FunctionDeclTree.

Evaluation subsystem

- ***Evaluator /* DSSEvaluator:** Exposes methods to perform operations upon DSS1 values, to interpret literals, allocate DSS objects, and perform conversions between DSS1 values and similar Java objects. (Note that Evaluator is an interface, whereas DSSEvaluator is the implementation; this distinction is used to permit and promote a decoupled implementation of the traversal subsystem by programming to an interface, as well as to facilitate stubbing and testing. The Evaluator interface is not strictly necessary and may be removed in a later milestone.)
    - ***DSSValue:*** An abstract class describing a value in a running DSS1 program. Primarily this exposes methods for the various operations (add, subtract, etc.) available under DSS1.
        - **DSSValueNull:** A null value in DSS1. Underlying Java object is an Object.
        - **DSSValueBool:** A boolean value in DSS1. Underlying Java object is a Boolean. (Exposes two constants, DSSBoolean.TRUE and DSSBoolean.FALSE, as a convenience.)
        - **DSSValueInt:** An integer value in DSS1. Underlying Java object is a Long.
        - **DSSValueFloat:** A floating-point value in DSS1. Underlying Java object is a Double.
        - **DSSValueTime:** A date & time value in DSS1. Underlying Java object is a Date.
        - **DSSValueList:** A list value in DSS1. Underlying Java object is a List<DSSValue>
        - **DSSValueObject:** An object in DSS1. Underlying Java object is a Map<String, DSSValue>.
        - **DSSValueString:** A string of text in DSS1. Underlying Java object is String.

Each instance of a DSSValue contains three elements of the DSS1 value triplet:

- *Value:* Stored in an appropriate field per type, interacted with via exposed methods.
- *Type:* Corresponds to the value's Java type identity (DSSValueString.class, for instance.)
- *Time of observation:* Stored as a DSSValueDate field. May be null for values that aren't read from encounters.

Intrinsic libraries

- ***DSSLibrary*:** Defines an interface for delivering or generating intrinsic functions in related groupings. Each function is returned in a map where the name should be used to call the function from a DSS program, and the function object is used as a Java object that extends the DSSFunction. An ExecutionContext is also available within the interpreter maintaining variables and certain functions. The following definitions in this document include the intrinsic functions belonging to DSSLibrary.
- ***AnnotatedDSSLibrary*:** The programmer annotates methods and arguments in a subclass of AnnotatedDSSLibrary which uses reflection to read these annotations and generate DSSFunctions.
  Conversions between DSSValue types and Java types are handled for return values and arguments by AnnotatedDSSLibrary, allowing the programmer to write plain Java code. Type conversions include:

DSS Type	*Java Type*
integer	long
float	double
boolean	boolean
date	java.util.Date
string	java.lang.String
list	java.util.List<DSSValue>
object	java.util.Map<java.lang.String, DSSValue>

- **ListLibrary:** Included in ListLibrary are functions that help sort and organize items.

  merge(list1, list2) – list1 or list2 could be a single item; this returns a new list, sorted in chronological order
  sortTime(list) – returns a list sorted based on *time* of items (from oldest to newest)
  sortData(list) – returns a list sorted based on *values* of items
  last(list) – return the last item in the list
  first(list) – return the first item in the list

- **DSSLengthAndWithinLibrary:** The length function simply returns either the list or the string, and the within function checks if the first item of the list is between the second and the third items.

  within(v,a,b) – returns true if v is between a and b, inclusive; e.g., within(3,2,5) is true
  length(v) – length of either list or string

- **IsLibrary:** A class that includes the various “is” intrinsics for type checking.

  isString(var)
  isInt(var)
  isFloat(var)
  isBoolean(var)
  isDate(var)
  isObject(var)
  isList(var)

- **DSSDateLibrary:** All intrinsic functions that are time oriented. Dates and times are possible components of any DSSValue. Date intrinsics take in all parameters as DSSValues and performs time comparison and return time values.

  currenttime() – return current time; e.g., Tue Nov 06 10:33:56 PST 2012
  recentTimeItem(list) – return the item with the most recent time
  oldestTimeItem(list) – return the item with the oldest time
  before(time1,time2) – return true if time1 is before time2
  time(v) – return time associated with v
  addDays(time,numDays) - return a new time based on numDays
  addMonths(time,numMonths) - return a new time based on numMonths

- **ReadLibrary:** Intrinsics for reading observation data for patients.

  read(patientId,conceptName) - returns a list of observations from db; consider lab vs other encounters; only retrieve items not voided
  readInitialEncounter(patientId,conceptName) - returns a list of observations from db, only from the **initial** encounter
  readLatestEncounter(patientId,conceptName) - returns a list of observations from db, only from the **latest** encounter

  The read functions retrieve a list of observations associated with a patient. The first parameter of the functions, patientId, is a numeric identifier unique to each patient. The second parameter, conceptName, is the word or phrase used by the OpenMRS dictionary to refer to a concept.
  The three functions are nearly identical, save for one difference: while read() returns a list containing all observations that match the function parameters, readInitialEncounter() and readLatestEncounter() filter out results based on the timestamp of the observations. Calling readInitialEncounter() retrieves only the observations from the patient's earliest encounter on record, while readLatestEncounter() retrieves only the observations from the patient's most recent encounter on record.
  When the functions are called, they retrieve a list of all encounters associated with patientId from the OpenMRS database. The functions iterate through these lists, and in the case of readInitialEncounter() and readLatestEncounter(), the timestamp for each encounter is checked. If the timestamp does not meet the criteria, the encounter is discarded. Once an encounter has been verified as valid the function shall retrieve all observations associated with the encounter. Each observation shall have its concept name checked against conceptName, and matches are added to the list of observations that each function shall return.
  Observations consist of three pieces of data: the value of the observation, the data type of the observation value, and the time of the observation. Internally, observations are represented as DSSValue objects, which store the value of the observation and the time of the observation. The data type of the observation value is stored as part of the DSSValue class type itself. Both the time and data type of the observation can be retrieved using the time() and type check intrinsics, respectively.

Sample Output:

For testing purposes, target “patientSummary” outputs to temporary patient dashboard tab “Rules”.

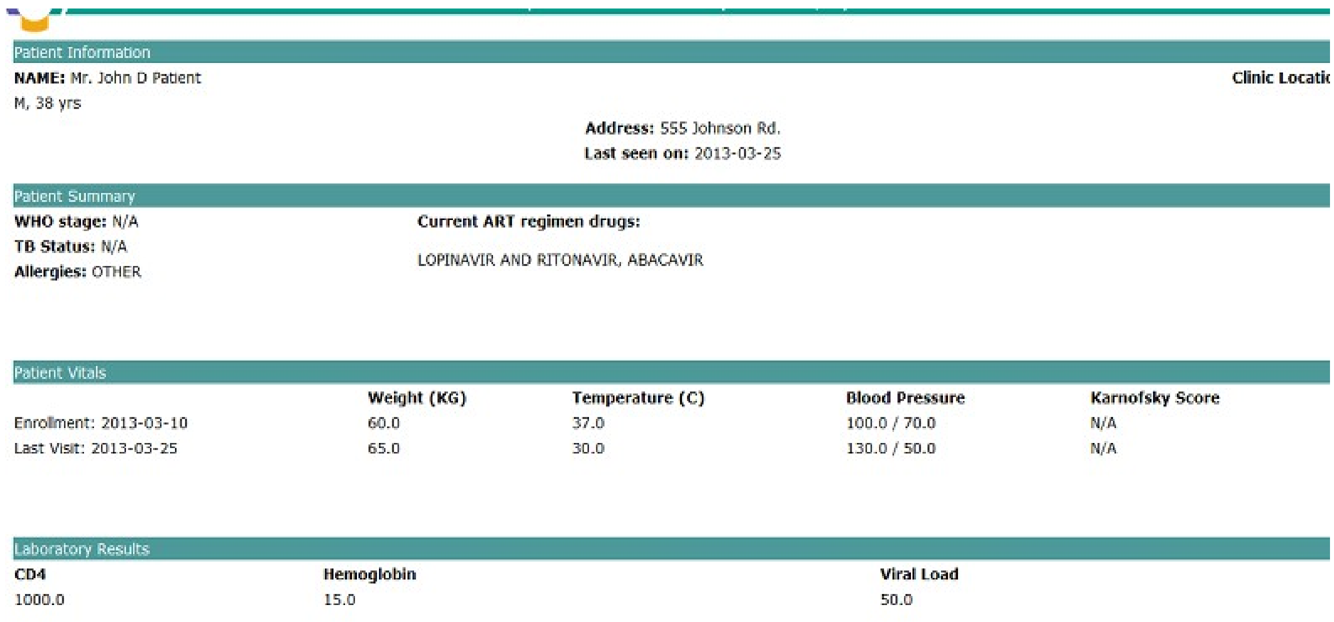
Rule Name: basicTest	Result
`program`	
`// BASIC TESTING OF INTRINSICS`	
`{`	
`      b     := true`	
`      lst   := {1,2,3,4,5}`	
`      time  := currenttime()`	
`      time2 := addDays(time, 7)`	
`      // Utilizing DSSRead functions`	
`      wt    := read(patientId, "WEIGHT (KG)")`	
`      // isLibrary`	
`      alert("patientSummary", isInt(1234))`	`True`
`      alert("patientSummary", isString("hello"))`	`True`
`      alert("patientSummary", isFloat(12.345))`	`True`
`      alert("patientSummary", isBoolean(b))`	`True`
`      alert("patientSummary", isList(lst))`	`True`
`      alert("patientSummary", isDate(time))`	`True`
`      // DSSListLibrary`	`{60.0, 60.0, 30.0, 30.0,`
`      alert("patientSummary", merge(wt, wt))`	`65.0, 65.0}`
`      alert("patientSummary", sortTime(wt))`	`{60.0, 30.0, 65.0}`
`      alert("patientSummary", sortData({1,10,4,15}))`	`{1, 4, 10, 15}`
`      alert("patientSummary", last(wt))`	`60.0`
`      alert("patientSummary", first(wt))`	`65.0`
`      // DSSDateLibrary`	
`      alert("patientSummary", addDays(time, 14))`	`Fri May 10 19:26:55 PDT 2013`
`      alert("patientSummary", addMonths(time, 2))`	`Wed Jun 26 19:26:55 PDT 2013`
`      alert("patientSummary", before(time, time2))`	`True`
`      alert("patientSummary", recentTimeItem(wt))`	`65.0`
`      alert("patientSummary", oldestTimeItem(wt))`	`60.0`
`      alert("patientSummary",`	`2013-03-25 00:00:00.0`
`time(recentTimeItem(wt)))`	
`      // Misc`	
`      alert("patientSummary", within(40,20,60))`	`True`
`      alert("patientSummary",`	`5`
`length({10,20,30,40,50}))`	
`}`	

Rule Name: blood pressure	Result

```
program

function bloodpressure()
{
      systolic := last(read(patientId, "SYSTOLIC
BLOOD PRESSURE"))
      diastolic := last(read(patientId, "DIASTOLIC
BLOOD PRESSURE"))
      alert("patientSummary", "Systolic: " +
systolic)
      alert("patientSummary", "Diastolic: " +
diastolic)
      check(systolic, diastolic)
}

function check(sys, dias)
{
      if ((sys < 120) | (dias < 80)) then
      {
            alert("patientSummary",
               "Normal Blood Pressure")
      }
      elsif (within(sys, 120, 139) |
              within(dias,    80,89)) then
      {
            alert("patientSummary",
               "Prehypertension")
      }
      elsif (within(sys, 140,159) |
              within(dias, 90, 99)) then
      {
            alert("patientSummary",
               "Stage 1 Hypertension.")
      }
      else
      {
            alert("patientSummary",
               "Stage 2 Hypertension.")
      }
}

{
      bloodpressure()
}
```
	Systolic: 100.0
	
	Diastolic: 70.0
	
	Normal Blood Pressure

Rule Name: random	Result
`program`	
`{`	
`      count := 0`	
`      date  := currenttime()`	
`      month := currenttime()`	
`      while(count < 12)`	
`      {`	
`            if (count == 10) then`	
`            {`	
`                alert("patientSummary",`	`Wed Feb 26 19:26:55 PST 2014`
`                   addMonths(date, count))`	
`            }`	
`            count := count + 10`	
`      }`	
`      list := read(patientId, "TEMPERATURE (C)")`	
`      wt := readLatestEncounter(patientId, "WEIGHT`	
`(KG)")`	
`      if (before(time(oldestTimeItem(list)),`	
`              time(first(wt))))then`	
`      {`	
`            alert("patientSummary",`	
`               "Number of observation of temperature`	`Number of observation of`
`               on record = " + length(list))`	`temperature on record = 3`
`      }`	
`      if (isList(wt)) then`	
`      { alert("patientSummary", merge(list, wt)) }`	`{37.0, 40.0, 30.0, 65.0}`
`}`	

Screenshots:



Patient Information

NAME: Mr. John D Patient — **Clinic Locatic**

M, 38 yrs

Address: 555 Johnson Rd.

Last seen on: 2013-03-25

Patient Summary

WHO stage: N/A

TB Status: N/A

Allergies: OTHER

Current ART regimen drugs:

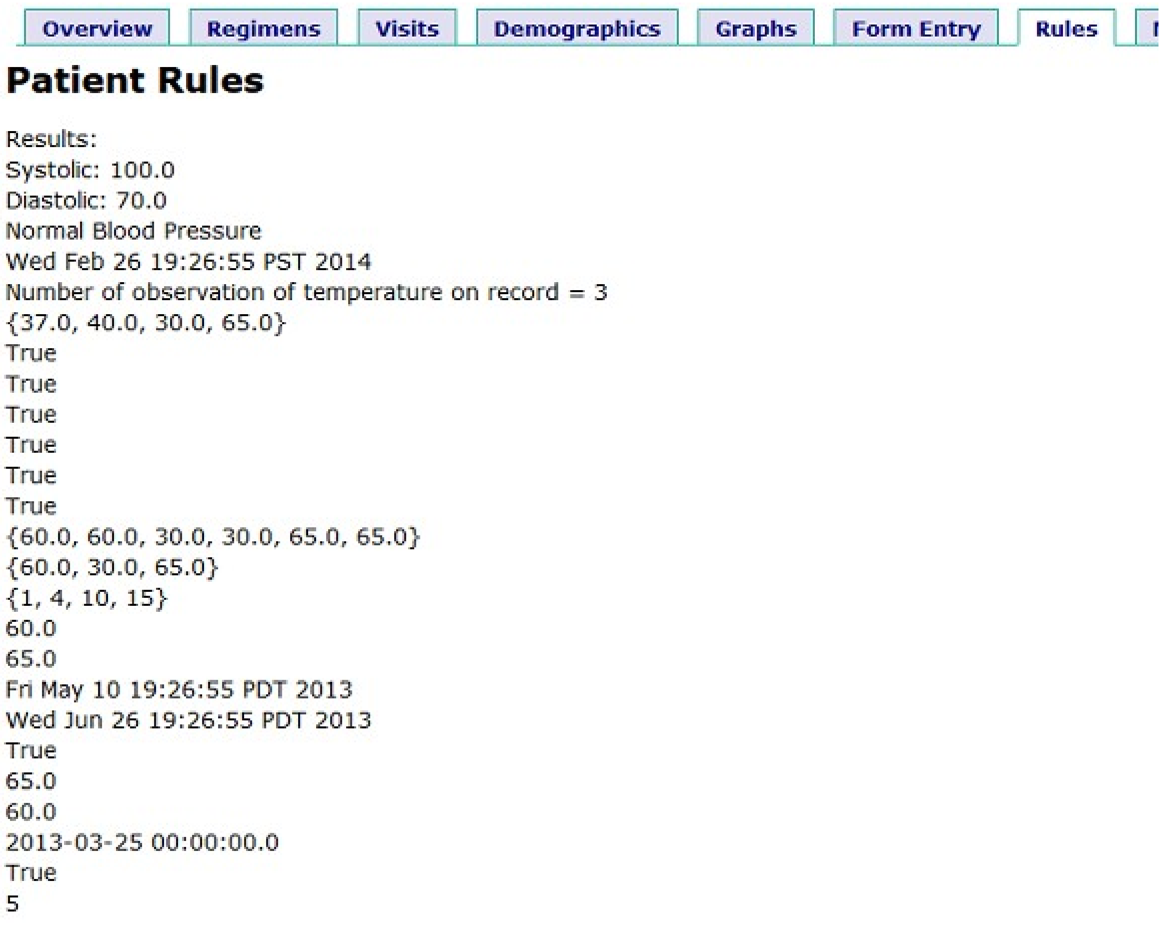
LOPINAVIR AND RITONAVIR, ABACAVIR

Patient Vitals

	Weight (KG)	Temperature (C)	Blood Pressure	Karnofsky Score
Enrollment: 2013-03-10	60.0	37.0	100.0 / 70.0	N/A
Last Visit: 2013-03-25	65.0	30.0	130.0 / 50.0	N/A

Laboratory Results

CD4	Hemoglobin	Viral Load
1000.0	15.0	50.0



Patient Rules

```
Results:
Systolic: 100.0
Diastolic: 70.0
Normal Blood Pressure
Wed Feb 26 19:26:55 PST 2014
Number of observation of temperature on record = 3
{37.0, 40.0, 30.0, 65.0}
True
True
True
True
True
True
{60.0, 60.0, 30.0, 30.0, 65.0, 65.0}
{60.0, 30.0, 65.0}
{1, 4, 10, 15}
60.0
65.0
Fri May 10 19:26:55 PDT 2013
Wed Jun 26 19:26:55 PDT 2013
True
65.0
60.0
2013-03-25 00:00:00.0
True
5
```