

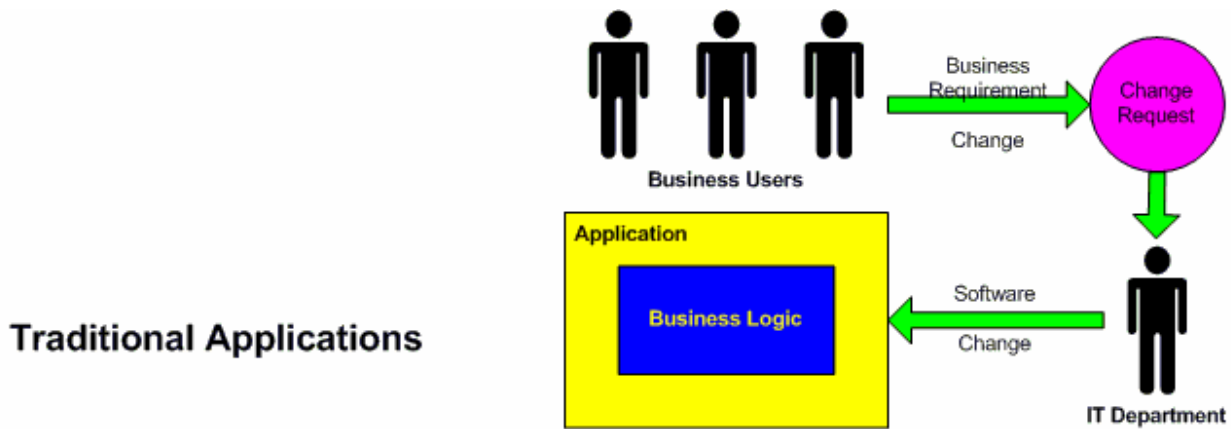
rules@work® Toolkit

Introduction

Business system developers have been increasingly struggling to keep pace with the business dynamics that are changing on a daily basis. The changing regulations, policies, marketing promotions and therefore business rules have also been the driving force behind software system changes. Traditionally, software developers used to hard-code all business rules and decisions within their program. In effect, IT departments have been increasingly occupied with the task of modifying and maintaining their work/programs to match the previously mentioned business rules. Overhead for designing, coding, testing and maintaining hard-coded business rules is becoming increasingly more expensive.

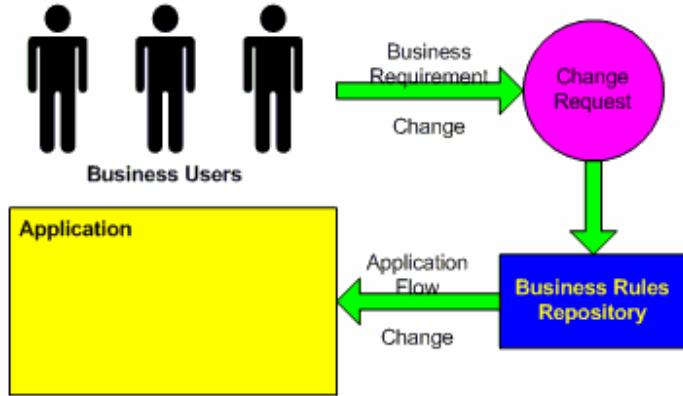
Recently, a new breed of tools has emerged resolving this globally acknowledged problem, allowing business experts and users to customize the way systems work through the definition of external business rules. In rules-based applications, all of the business logic is located outside of the compiled application in a separate rule engine, therefore making the process of changing business logic much easier.

Traditional Applications versus Rule-based Applications



In traditional applications most, if not all, of the business rules are located in compiled code. This makes the process of changing business logic lengthy and cumbersome. The business users request a change to the business logic, the change request gets forwarded to a change control board, this is evaluated by the board and approved, the IT department receives change request and does an impact study of what will be affected by change, the IT department completes changes and begins testing, the QA team completes testing, and then the change is implemented in production. In many cases, it might take months to complete even minor changes in business logic in a traditional application.

Rule Based Applications



In rules-based applications, all of the business logic is located outside of the compiled application in a separate rule engine. This rules engine contains the program flow directives that normally drive an application. This makes the process of changing business logic much easier. The business users request a change to the business logic, the change request gets forwarded to a change control board, this is evaluated by the board and approved, a business analyst or IT person implements the change and begins testing, the QA team completes testing, and then the change is implemented in production. In many cases, it might take only a few days to complete some changes in business logic in a rules based application. Rules-based application tend to be easier to maintain, and are maintainable at a lower cost because code does not have to be redesigned, retested, recompiled and redeployed for every minor change in business processes.



Health Insights FZ LLC proudly introduces its rules processing toolkit: *rules@work*®. The *rules@work*® toolkit incorporates the best of what rules-based engines have to offer, allowing the creation of business-specific rule engines, under which many knowledge bases can be created. Easier maintainability and customization of the systems and business rules is achieved with minimum effort, costs and time.

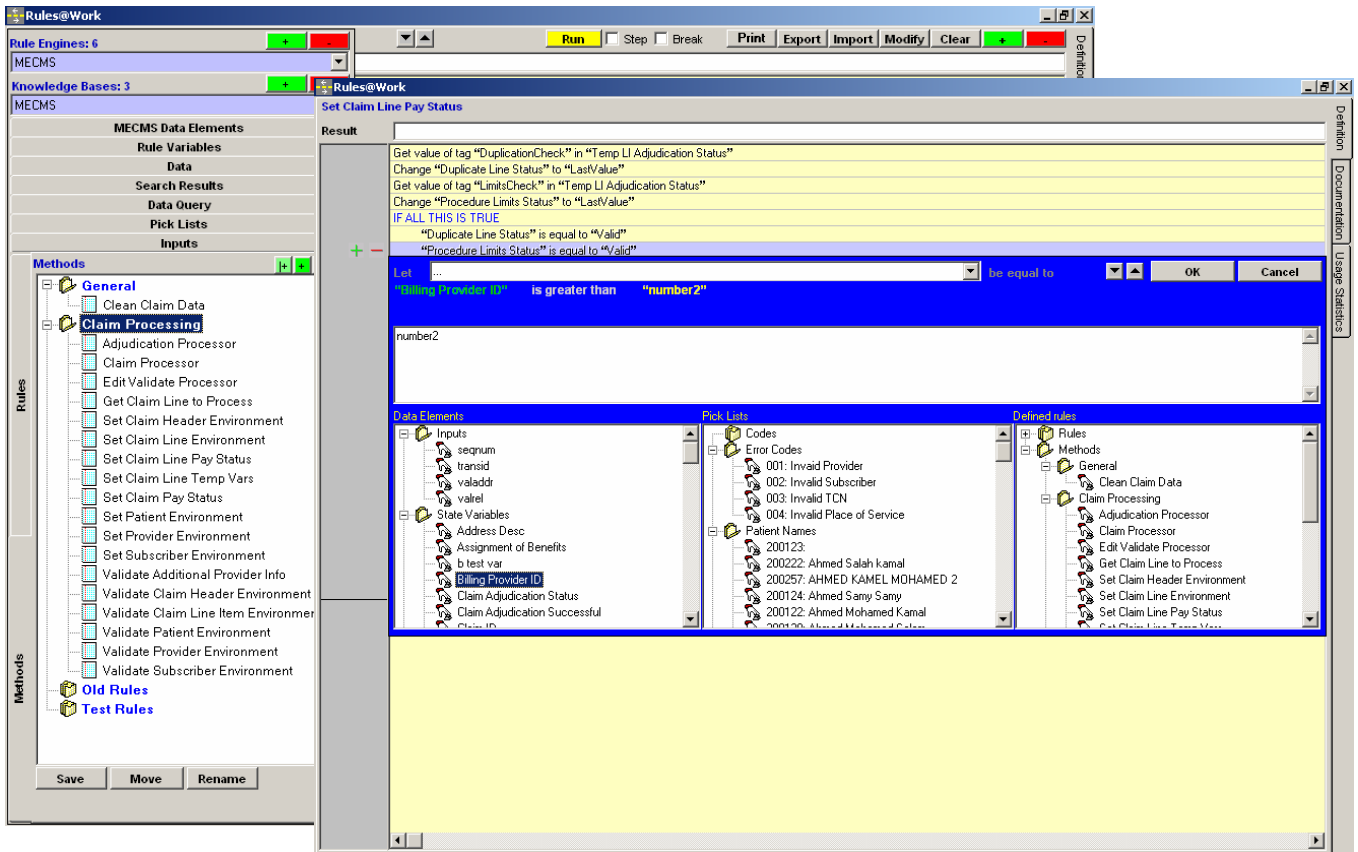
rules@work® toolkit comprises three components: the *rules@work*® *Composer*, the *rules@work*® *Processor* and the *rules@work*® *Service Administrator*.

All three components of the *rules@work*® toolkit allow for maximum performance, flexibility and extensibility of applications that utilize them.

rules@work® Composer

The composer is the integrated rule development environment in which users can create, test, document and version control rules and methods. It includes a comprehensive set of functions/interfaces that simplifies the task of rule creation to a simple point-and-click/fill-in-the-blanks operation. It supports the following features:

- Graphical User Interface to create, modify, and test rules
- English language syntax for ease of use
- Access to externally created objects to access/create information
- Point and click 'fill-in-the-blank' functionality for small learning curve on syntax
- Documentation and version control for QA
- Variable and rule/method usage statistics for change impact analysis
- Several execution methods for testing:
 - o Run – execute to completion without stopping
 - o Step – step through every individual rule line including nested rule/method calls
 - o Break – run to break point and pause execution
- Simulation of inputs from callers (Web pages/applications)
- Security control of rule engines, knowledge bases and databases



Rules@Work

Set Claim Line Pay Status

Run Step Break Print Export Import Modify Clear

Result

```

Get value of tag "DuplicationCheck" in "Temp LI Adjudication Status"
Change "Duplicate Line Status" to "LastValue"
Get value of tag "LimitsCheck" in "Temp LI Adjudication Status"
Change "Procedure Limits Status" to "LastValue"
IF ALL THIS IS TRUE
  "Duplicate Line Status" is equal to "Valid"
  "Procedure Limits Status" is equal to "Valid"
THEN
  / IF ANY OF THIS IS TRUE
  / "Service Location" = "12"
  / "Service Location" = "99"
  / "Service Location" = "11"
  /
  / THEN
  / Divide "Temp LI Charge Amount" by "2"
  / Change "Temp LI Charge Amount" to "LastValue"
  /
  / OTHERWISE
  /
  Add "Total Claim Pay Amount & Temp LI Charge Amount"
  Change "Total Claim Pay Amount" to "LastValue"
  Change "Temp LI Pay Status" to "True"
OTHERWISE
  Change "Temp LI Pay Status" to "False"
  Change "Claim Pay Status" to "False"
  
```

Definition Documentation Usage Statistics

Rules@Work

Set PRIME-CARE-EXCEPTION Switch

Author Version Date Activation Date Phase Out Date Status

Nasser Shehata 1 01/01/2002 01/01/2002 14/01/2002 Development Time Sensitive Check In

Version Changes

Initial Rule Version

Description

Sets the prime-care exception switch based on provider type, type of service and program code

Versi...	Date	Description	Author	Activation	Phase Out
1.7	16/08/2002	Real New version	Nasser Sheh...	22/06/2002	
1.6	02/01/2002	New version	Nasser Sheh...	16/05/2002	21/06/2002
1.5	02/01/2002	Testing Save Current	Nasser Sheh...	27/04/2002	15/05/2002
1.4	25/03/2002	Commented all first and third IFs	Nasser Sheh...	08/04/2002	26/04/2002
1.3	24/03/2002	Commented all first, second and third IFs	Nasser Sheh...	29/03/2002	07/04/2002
1.2	23/03/2002	Commented both first and second IFs	Nasser Sheh...	23/03/2002	28/03/2002
1.11	21/03/2002	Change to V 1.1	Nasser Sheh...	20/03/2002	22/03/2002
1.1	20/03/2002	Commented the second IF ALL	Nasser Sheh...	20/03/2002	20/03/2002
1.01	15/01/2002	Commented first IF ALL	Nasser Sheh...	15/01/2002	19/03/2002
1	01/01/2002	Initial Rule Version	Nasser Sheh...	01/01/2002	14/01/2002

Definition Documentation Usage Statistics

rules@work® Processor

The Processor is the runtime rule evaluator and executor. It runs as an unattended service on the application server. A simple calling sequence invokes the processor at the beginning of rule set processing. The processor then interacts through the rule syntax with all external support objects, examining their properties and executing methods in them depending on the logic previously defined in the Composer. After processing has been completed for the rule set, the final state of the processor is returned to the calling application/web page for analysis and further processing.

rules@work® Service Administrator

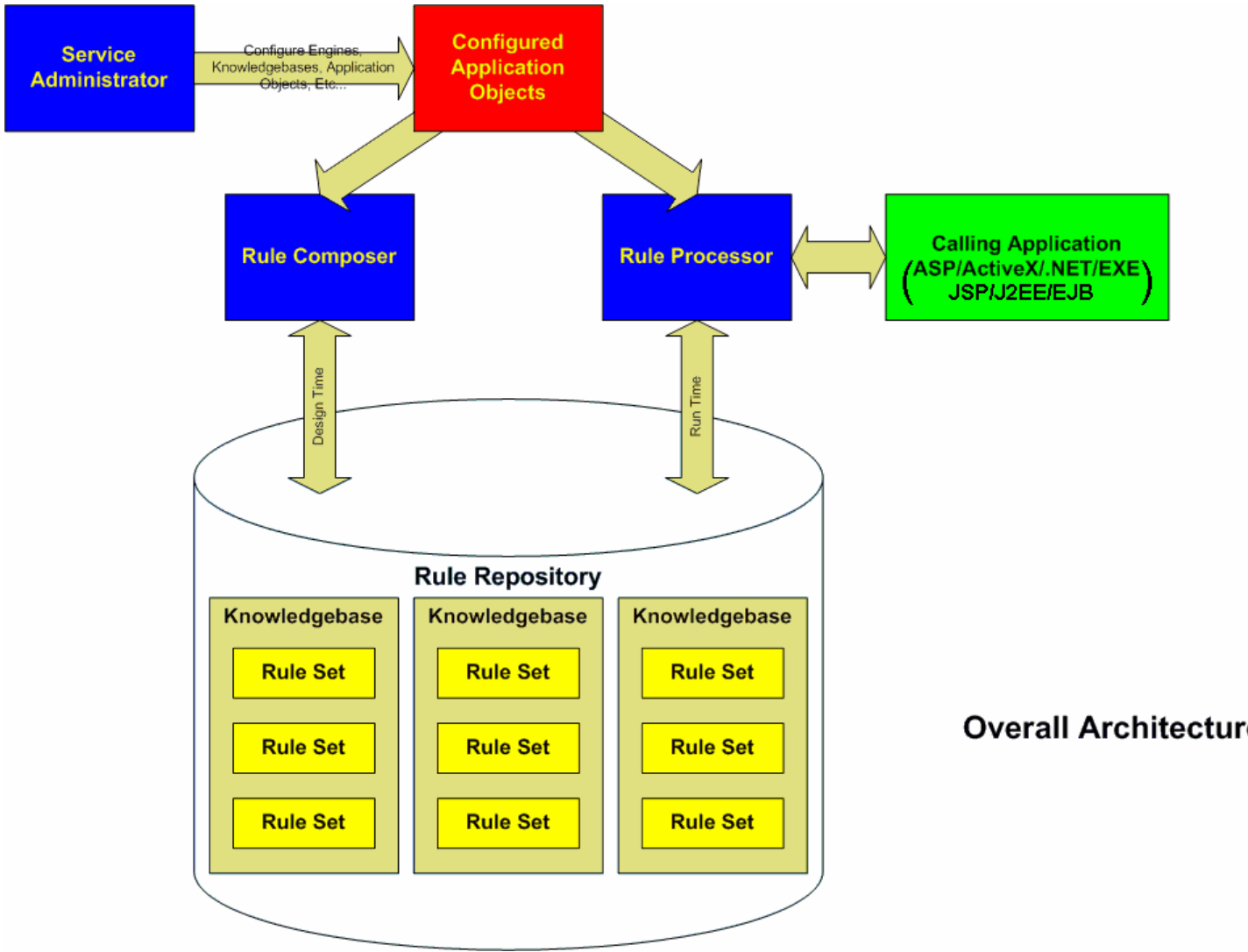
The Service Administrator is the component that manages users/groups and their access rights to rule engines and their knowledge bases. It is also responsible for the creation and management of rule engines and their knowledge bases, inputs, state variables, parametric database queries, pick lists, etc...

The Service Administrator is also responsible for registering external support objects (fact finder and doer objects) to the *rules@work®* database. Objects are reflected to the environment where their desired methods and public properties/fields can be aliased with English language expressions to enhance readability for business users.

Multiple data connections can be configured in the Service Administrator for use by rules/methods in the Composer, while creating/editing rules/methods, and at runtime by the Runtime Processor.

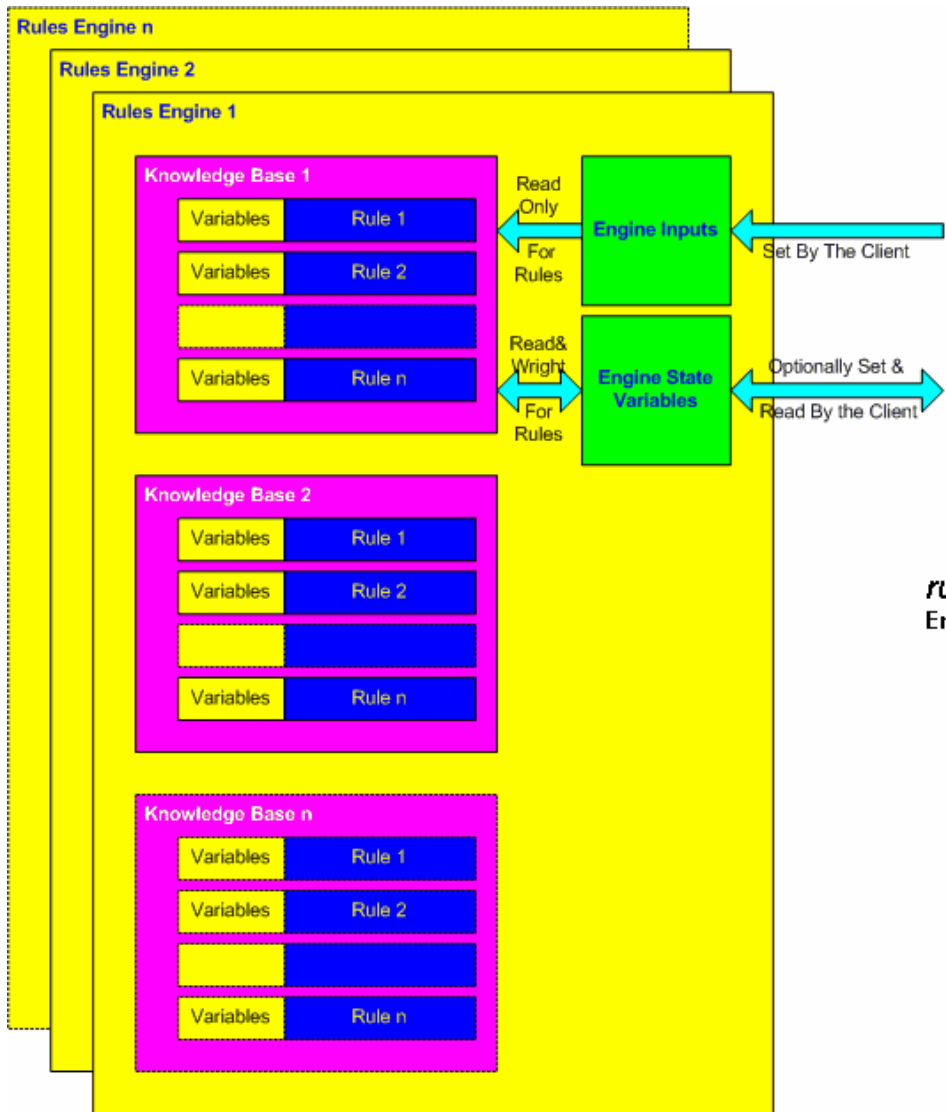
System Architecture

rules@work® acts as a repository of code that handles various business situations. Its role is to accumulate business knowledge in small manageable pieces that are called business rules. Because of today's complexity of business relationships, regulations and situations, rules that decide on a certain path of action may be based on several parameters. Such parameters are different from one domain of applications to the other.



rules@work® identifies an application domain as a Rule Engine. You can define multiple Rules Engines for as many applications as you have. Each engine can have multiple knowledge bases that serve diverse requirements. For example, a Rules Engine for a project-level application could have several Knowledge Bases: Data Conversion, Client Address

Maintenance, Payment Processing, etc. Each of these would have their own Rules and Methods, but would share the same Inputs and State Variables.



rules@work
Engine Concepts

Key Features

- Executes rules that control the business logic flow
- Can be invoked from different sources:
 - o Web Applications
 - o Desktop Applications
 - o *rules@work* Rule Composer
- Access to externally created objects to access/create information
- Maintains/returns rule set execution state variable information
- Modularity
- Database Independency.
- Scalability (Integration).
- Flexibility and ease of use.
- Extensibility.
- Management of multiple rule engines and knowledge bases at the same time to allow for grouping of rules for different applications.
- Testing/debugging environment allows users to 'run', 'step', and 'break' into rules execution to analyze function and results.
- Import, Export, and Print rules to text format for analysis by business experts.
- Support for:
 - o Flow Control Functions (If All, If Any, If None, Loops, Bypasses, Returns, Etc...)
 - o Nested Calls
 - o Constants, Global and Local Variables
 - o Dynamic variable creation at runtime
 - o Comparison Functions
 - o Math Functions
 - o String Functions
 - o Database Functions
 - o Parametric defined database queries with English aliases
 - o Application specific user definable pick lists for
 - o Data/Time Functions
 - o User Interaction Functions

rules@work® allows the creation of business-specific rule engines, under which many knowledge bases can be created. Each engine has its own set of inputs that are passed to the engine either programmatically or interactively for processing. Each engine also has its own set of State Variables, which are manipulated by the engine rules to return back a engine state at the end of a rule processing cycle. Each knowledge base within an engine can have its own Rules and Methods. Both Rules and Methods are callable from one another. Methods are functions that return a value (e.g., 'Calculate age at a point of time given the date of birth'). Rules are logic processing units that can call upon other rules. Both Rules and Methods syntax and functionality can be extended through the use of externally developed components/objects. Syntax for functions and statements in a rule is entered through a point and click graphical user interface in an English-like language. Even external functions can be given a custom English syntax for ease of use and familiarity to business users.

rules@work® also supports multiple database access from within the Rules and Methods. Database tables, views and columns can have aliases for better clarity for business users.



The *rules@work*® *Composer* is used at design time to create, modify and test Rules and Methods in a controlled environment. Inputs and State Variables can be set and changed manually by the user. Rule/Method execution can be stepped through for step-by-step testing and debugging. Results of each line in a rule is displayed and reflected within the Composer. Rules/Methods can have their own local variables and they can dynamically create and manipulate State Variables. Both Inputs and State Variables are passed between Rules and Methods for manipulation. The final set of State Variables are returned to the calling application for further processing.

Operation Overview

The **rules@work®** Composer/Processor can utilize the following in processing rules:

- **Inputs** - These are parameters that are passed when a Rule or Method is called. They are read-only (not changeable) by the Rules Engine.
- **State Variables** - These are global Variables available for all Rules and Methods to read and change.
- **Rule Variables** - These are local Variables available only for the specific Rule or Method that created them.
- **Rules/Methods** – Rules and Methods can be called from within each other or by outside applications. Typically, Methods would be created by programmer/analysts and would be regarded as low-level activities, whereas Knowledge Rules could be maintained by business analysts and control the business process.
- **Fact finder objects** - These are externally created Objects used to do the heavy lifting of database fetching.
- **Doer objects** - These are externally created Objects used to do the heavy lifting of database saving or heavy computations.

The **rules@work®** Runtime Processor is called or invoked by an application or web page. Inputs and a Rule ID or name are passed to the Engine by the calling application. These Inputs are used by the Rules@Work Engine to set Variables, pass to Rules, pass to Methods, and pass to external Objects. Fact finder objects use the Inputs and Variables passed to fetch data from the database or interact with other external support objects. Support objects can access the engine state variables which represent other support objects. The **rules@work®** Processor calls other Rules and Methods to make decisions based on call results to the support objects and changes in values of the state variables. The Processor can pass information to the Doer support objects to save data to a database or to further communicate with and interact with other external support objects. Once a rule set is completely processed, the **rules@work®** Runtime Processor then returns the result of the business process back to the calling application in the form of the current values of the engine's state variables. The calling application/component/web page can then examine the values for further processing.

Rules Engine Calling Sequence

