**GSTV Rule Engine Architecture**May 2016

The GSTV Rule Engine Architecture (GREA) is built on top of Nools (Javascript Rete-based Rule Engine) to accommodate applications with large number of rules by introducing the concept of Rule Modules. When writing large rule applications, the programmers often create groups of rules relying on the language provided “agenda-group”. In GREA, Rule Module provides additional supports to allow quick contruction of rule applications and reuse of existing rule modules via composition, as well as running and testing rule modules individually or as a group.

In GREA, Rule Module contains a set of rules. It has accesss to the Global Context in addition to a Local Context of its own. Global Context contains the Nools engine and session as well as shared resources to all rule modules. Local Context contains local information to a particular rule module such as rule files and sequence of focused agenda. Rule Modules can also contain other rule modules. This concept allows the creation of rule module tree structures, which is an important device in rule programming. Tree structure also lends itself to breath-first of depth-first problem solving techniques. Figure 1 is the example of a tree of rule modules.

globalContext

localContext

localContext

Figure 1: Example of rule groups in a tree structure

A rule module is orgranized into a single folder, containing at least 2 sub-folders named “config” and “rules” and an index.js file.

**Global Context:** is a function to create a global context object that is accessible from all rule modules. It contains Nools flow and session objects as well as resources needed for nools operations such as agregated rules, initial facts and classes. To create a global context, one needs to import it from the globalContext module, then run

import initGlobalContext from './globalContext';   
this.globalContext = initGlobalContext();

**Local Context:** is a class that encapsulates information particular to a rule module such as name, the rule file, and focus agenda.

**Rule Module:** is the main vehicle for the organization of rules. Each rule module can contain its own set of rules, as well as other rule modules. The nested rule module containment concept allows construction of hierarchial tree of rule modules. A rule module therefore can act as an application by itself.

Rules components are parts of rules that can be utilized inside other rules. For example, if you have a custom action that you want to use in multiple rules, a rule component made up of the custom action can be created then simply added to each rule. When the "Rules Scheduler" module is enabled, components can be scheduled to run at a future time. This is a common use case for rules components.

**Agenda Control for Rule Module:** The Agenda is a *Rete* feature. It maintains set of rules that are able to execute, its job is to schedule that execution in a deterministic order.

During actions on the RuleRuntime, rules may become fully matched and eligible for execution; a single Rule Runtime Action can result in multiple eligible rules. When a rule is fully matched a Rule Match is created, referencing the rule and the matched facts, and placed onto the Agenda. The Agenda controls the execution order of these Matches using a Conflict Resolution strategy.

The engine cycles repeatedly through two phases: Rule Runtime Actions. This is where most of the work takes place, either in the Consequence (the RHS itself) or the main Java application process. Once the Consequence has finished or the main Java application process calls fireAllRules() the engine switches to the Agenda Evaluation phase. Agenda Evaluation. This attempts to select a rule to fire. If no rule is found it exits, otherwise it fires the found rule, switching the phase back to Rule Runtime Actions.

**Server:** is a singleton acting as the starting point of the engine always return the initialized application.

**App:** is the application wrapper of a rule module to also provide opportunity to integrate application-specific logic on top of a rule module.