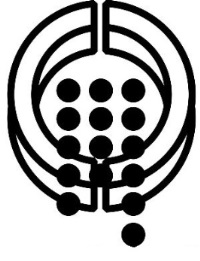


****

**شرکت صنایع الکترونیک ایران بنیاد نخبگان ستاد کل  
 شرکت ایز ایران**

**گزارش میانی فعالیت دوره ضرورت نخبگان بخش دفاع**

**عنوان**

**پیاده سازی و مجتمع سازی ماژول مدیریت قوانین کسب و کار در زیر ساخت تولید نرم افزار رایتن**

استاد راهنما  
جواد پوردیلمی

**مجری**

سعید شکوهی

**آبان 1394**

**فهرست مطالب**

|  |  |
| --- | --- |
| عنوان مطالب | شماره صفحه |

چکیده ............................................................................................................................................................................1

[1 ) فصل اول : كليـات مفاهیم موتورهای قوانین کسب و کار 8](#_Toc434618164)

[1-1 ) مقدمه 8](#_Toc434618165)

[1-2 )قوانین کسب و کار... 9](#_Toc434618166)

[1-3 )مقدمه ای بر موتور قوانین کسب و کار....... 10](#_Toc434618167)

[1-4 ) تعریف موتور قوانین کسب و کار...... 10](#_Toc434618168)

[1-4-1 ) پیشینه موتور قوانین کسب و کار 10](#_Toc434618169)

[1-5 ) علل استفاده از موتور قوانین کسب و کار…. 16](#_Toc434618170)

[1-5-1 ) مزیت های استفاده از موتور قوانین 16](#_Toc434618171)

[1-5-2 ) شرایطی که باید از موتورهای قوانین استفاده کرد …. 19](#_Toc434618172)

[1-6 ) شرایطی که نباید از موتور قوانین استفاده کرد….. 21](#_Toc434618173)

[1-7 ) ویژگی های سیستم های مدیریت قوانین کسب وکار..... 22](#_Toc434618174)

[1-7-1 ) کامپوننت ها و ویژگی های فنی سیستم مدیریت قوانین کسب و کار (BRMS) 23](#_Toc434618175)

[1-7-2 ) قوانین 25](#_Toc434618176)

[1-7-3 ) قالب های قوانین 25](#_Toc434618177)

[1-7-4 ) چک کردن نحوی قوانین 25](#_Toc434618178)

[1-7-5 ) رویه ها و الگوریتم ها 26](#_Toc434618179)

[1-7-6 ) جریان قوانین 26](#_Toc434618180)

[1-7-7 ) جداول و درخت تصمیم گیری 26](#_Toc434618181)

[1-7-8 ) استنتاج 27](#_Toc434618182)

[1-7-9 ) عدم قطعیت و شرح دهی 28](#_Toc434618183)

[2 )فصل دوم : بررسی سیستم های مدیریت قوانین کسب و کار موجود 30](#_Toc434618184)

[2-1 ) مقدمه 30](#_Toc434618185)

[2-2 ) معرفی سیستم های مدیریت قوانین موجود…. 31](#_Toc434618186)

[2-2-1 )Drools 31](#_Toc434618187)

[2-2-2 )Jess 32](#_Toc434618188)

[2-2-3 )Hammurapi Rules 33](#_Toc434618189)

[2-2-4 )InfoSapient 34](#_Toc434618190)

[2-2-5 )JLisa 35](#_Toc434618191)

[2-2-6 )Mandarax 35](#_Toc434618192)

[2-2-7 )OpenLexicon 36](#_Toc434618193)

[2-2-8 )OpenRules 36](#_Toc434618194)

[2-2-9 )Jamacho (morendo) 37](#_Toc434618195)

[2-2-10 )JRuleEngine 37](#_Toc434618196)

[2-3 ) مقایسه موتور قوانین کسب و کار…. 39](#_Toc434618197)

فهرست شکل ها و نمودارها

[شکل 1- نمایش سطح بالایی از موتور قوانین … 13](#_Toc434618421)

[شکل 2- روش استنتاج زنجیره ای رو به جلو 14](#_Toc434618422)

[شکل 3- روش استنتاج زنجیره رو به عقب 15](#_Toc434618423)

[شکل 4- معماری سطح بالا سیستم مدیریت قوانین کسب و کار 24](file:///C:\Users\saeed\Desktop\drools\documenrt\گزارش%20میانی.docx#_Toc434618424)

[شکل 5- معماری سیستم مدیریت قوانین Drools 32](#_Toc434618425)

[شکل 6 - معماری کامپوننت های Hammurapi Rules 34](#_Toc434618426)

[شکل 7 - معماری سیستم مدیریت قوانین OpenRules 37](#_Toc434618427)

**چکیده**

ازآنجا که اغلب نرم افزار های مورد نیاز وزارت دفاع ، نرم فزار های مبتنی بر قانون می باشند و حجم قوانین منطقی آنها زیاد است . از سوی دیگر، جزئیات این قوانین ثابت نیست و با توجه به شرایط موجود تغییر می­کند . اعمال قوانین در نرم افزار با توجه به حجم زیاد قوانین باعث ایجاد کدهای پیچیده ای میشود که اغلب نگهداری این کدها در بلند مدت بسیار دشوار خواهد بود . و مهمتر از آن در حالتی که قوانین در زمان اجرای نرم افزار تغییر میکنند نیاز است که توسط برنامه نویس در کد تغییرات اعمال شود و بعد از کامپایل مجدد در برنامه قابل مشاهده خواهد بود . در حالی که نیازمندی این است که کاربر نهایی آموزش دیده بتواند در حالت اجرا قوانین را تغییر دهد و بدون متوقف شدن نرم افزار اثر تغییرات را در برنامه ببیند .

فصل اول

كليـات مفاهیم موتورهای قوانین کسب و کار

# فصل اول : كليـات مفاهیم موتورهای قوانین کسب و کار

## مقدمه

کسب و کار بر اساس قوانین اجرا می شوند . آنها فرآیندهای کسب و کار را تعریف می کنند،و توصیف می کنند که اگر چیزی درست باشد یا غلط باشد چه اتفاقی خواهد افتاد . مانند اینکه "*همه مشتریان با رتبه طلایی 10 درصد تخفیف بگیرند و خدمات خاصی دریافت کنند* " . قوانین کسب و کار بخش­هایی از سیستم های پشتیبانی کننده از تصمیم گیری[[1]](#footnote-1) هستند که هر فرآیند کسب و کار را پی ریزی می کنند .

می توان قوانین را در منطق کسب و کار برنامه کاربردی پیاده سازی کرد. بدین صورت که اگر بخواهید هر قانون را بصورت شرط های تودرتو بنویسید .آنگاه بعد از مدتی به کدهای اسپاگتی[[2]](#footnote-2) تبدیل خواهند شد . از سوی دیگر در طول عمر[[3]](#footnote-3) نرم افزار قوانین کسب و کار تغییر خواهند کرد . تغییر و خطایابی[[4]](#footnote-4) هر شرط و قانون، پیچیدگی و زمان توسعه وتست نرم افزار را افزایش خواهد داد و در عین حال صاحب فرآیند کسب و کار منتظر اعمال آخرین تغییرات قوانین کسب و خواهد ماند .

درست این است که شما کد های قوانین را خودتان ننویسید . راحت تر است که از یک موتورقوانین کسب و کار برای رسیدگی ، مدیریت و سازماندهی قوانین استفاده کنید . موتورهای قوانین اعلانی [[5]](#footnote-5) هستند و این اجازه را به شما می دهند که گروه های قوانین را که رویدادها و سرویس ها را راه اندازه می کنند . بدون سرزدن به برنامه نویسی رویه ای، به سرعت در کنار هم قرار دهید. همچنین اگر موتور قوانین کسب وکار امکانات و ابزارهای مناسبی برای ایجاد و تغییر قوانین فراهم کند . می­توان مسئولیت ایجاد و تغییر قوانین کسب و کار به صاحبین کسب کار واگذار کرد. این بدین معناست که تیم توسعه، بعد از هر تغییر قوانین کسب و کار نیاز به تغییر کد های برنامه کاربردی ندارند.

## قوانین کسب و کار[[6]](#footnote-6)

قانون کسب و کار، یک قاعده برای یک کسب و کار، شرکت و یا یک سازمان است. قانون کسب و کار، قانونی است که برخی از جنبه های کسب و کار را تعریف و یا محدود کرده و همیشه به حل درست یا نادرست برخی از مسائل در این حوزه ها می انجامد. قوانین کسب و کار برای اثبات ساختار کسب و کار و یا برای کنترل یا تحت تاثیر قرار دادن رفتار کسب و کار در نظر گرفته می شود. قوانین کسب و کار عملیات ها، تعاریف و محدودیت هایی را که در یک سازمان اعمال می شود را توضیح می دهد. قوانین کسب و کار می تواند در مورد افراد، فرآیندها[[7]](#footnote-7)، رفتار سازمانی و سیستم های محاسباتی در سازمان اعمال شود و برای کمک به سازمان در دستیابی به اهداف خود در جای خود مورد استفاده قرار گیرد. به عنوان مثال یک قانون کسب و کار ممکن است اینگونه تصریح کند که نیازی به بررسی اعتبار در مورد مشتریان خاص نیست. نمونه های دیگر از قوانین کسب و کار عبارتند از: نیاز به یک عامل اجاره برای نپذیرفتن یک مستاجر اجاره ای در صورت پایین بودن رتبه بندی اعتباری آن و یا نیاز به عوامل شرکت برای استفاده از تعدادی از تامین کنندگان ترجیحی و برنامه های عرضه. در حالی که یک قانون کسب و کار ممکن است رسمی یا حتی نانوشته باشد نوشتن قواعد روشن و اطمینان از اینکه این قوانین با سایر قوانین در تضاد نیست خود یک فعالیت ارزشمند محسوب می شود. قوانین کسب و کار جزئیات اقدامات را به یک سازمان معرفی می کند در حالی که استراتژی از چگونگی تمرکز کسب و کار در سطح کلان برای بهینه سازی نتایج صحبت می کند. قوانین کسب و کار در مورد نحوه تبدیل یک استراتژی به عمل رهنمودهای دقیق ارائه می کنند. قوانین کسب و کار برای یک سازمان در مورد آگاهی های سازمانی صحبت می کند. یک سازمان ممکن است بصورت فعالانه برای شرح شیوه های کسب و کار خود، اقدام به تولید یک پایگاه اطلاعاتی از قوانین کسب و کار نماید.

## مقدمه ای بر موتور قوانین کسب و کار[[8]](#footnote-8)

موتورهای قوانین با سنجش و ارزیابی حقایق [[9]](#footnote-9) و استفاده از نتایج آنها حقایق جدیدی را ایجاد می­کنند. بر اساس این تعریف بزرگترین مشکل پیش رو برای استقرار موتور قوانین کسب و کار ، پیدا کردن ابزارو تکنولوژی مناسب نیست بلکه تعریف صحیح قوانین است . این مهم است که همه قوانینی که هم اکنون در حال استفاده هستند منعکس شوند و انتقال قوانین کار ساده­ای نیست . تجربه نشان داده است که قوانین به همان اندازه که صریح و روشن هستند به همان اندازه ضمنی و پنهان نیز می باشند.

## تعریف موتور قوانین کسب و کار

### پیشینه موتور قوانین کسب و کار

هوش مصنوعی[[10]](#footnote-10) حوزه گسترده­ای از تحقیقات است که بر روی "*فکر کردن کامپیوتر مانند انسان* " تمرکز دارد. و شامل بخش های مانند شبکه های عصبی[[11]](#footnote-11)، الگوریتم ژنتیک[[12]](#footnote-12)، درخت تصمیم گیری و سیستم های خبره[[13]](#footnote-13) می­باشد. نمایش دانش [[14]](#footnote-14) بخشی از حوزه هوش مصنوعی است که با اینکه چگونه دانش بازنمایی و دستکاری می­شود مرتبط است. سیستم های خبره نیز از سیستم های مبتنی بر دانش برای تسهیل کدکردن دانش به پایگاه دانشی که جهت اسنتاج قابل استفاده باشد،بهره می­برد. سیستم های خبره نیز بعنوان سیستم های مبتنی بر پایگاه دانش شناخته می­شوند و سیستم های خبره مبتنی بر پایگاه دانش نیز به عنوان هوش مصنوعی کابردی در نظر گرفته می­شوند . فرآیند توسعه با یک سیستم خبره مهندسی دانش گفته می­شود.

سیستم های مدیریت قوانین کسب و کار [[15]](#footnote-15) امکانات افزوده ای را بر روی موتور قوانین همه منظوره با فراهم کردن امکان ایجاد،مدیریت ، توسعه ترکیب و آنالیز قوانین برای کاربران کسب و کار می سازند . که به رسمی کردن نقش قوانین موتور قوانین در فرآیند تجاری کمک می کنند .

موتور قوانین به هر سیستمی گفته می شود که به هر شکلی از قوانین استفاده می­کندتابر روی داده ها اعمال می­شود و خروجی تولید می­کند. که شامل سیستم های ساده مانند اعتبارسنجی و موتورهای عبارت های پویا می­شود .

یک سیستم قوانین تولید زمانی کامل می شود که بر روی بازنمایی دانش جهت ارائه گزاره ای و منطق مرتبه اول در یک روش مختصر ، اعلانی و نا مبهم تمرکز داشته باشد . هسته متفکر یک سیستم قوانین تولید[[16]](#footnote-16) ، یک موتور استنتاج حقایق و داده ها را با قوانین تولید مطابقت می دهد و عملیات متناظر را انجام می دهد. یک قانون تولید از ساختار دوبخشی تشکیل شده است که از منطق مرتبه اول برای استنتاج بر روی بازنمایی دانش استفاده می کند .

when

    <conditions>

then

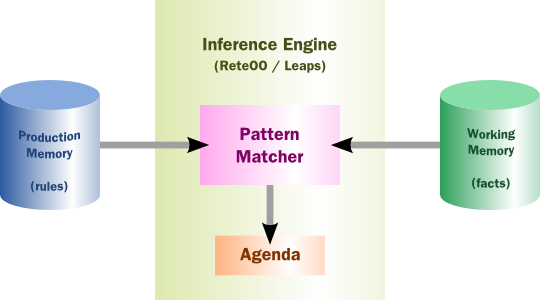
    <actions>;

فرآیند انطباق[[17]](#footnote-17) بر روی حقایق جدید یا موجود با قوانین تولید را الگوی انطباق[[18]](#footnote-18) می گویند . که توسط موتور قوانین انجام می شود . تعدادی الگوریتم برای الگوی انطباق وجود دارد و توسط موتورهای استنتاج استفاده میشوند:

* Linear
* Rete
* Treat
* Leaps

به عنوان مثال موتور قوانین Drools از الگوریتم Rete استفاده می کند . الگوریتم leaps قبلا استفاده میشده اما بدلیل غیر قابل نگهداری بودن آن استفاده نمی شود . پیاده سازی الگوریتم Rete توسط Drools به نام ReteOO شناخته می شود .که نشان می دهد که این الگوریتم برای سیستم های مبتنی بر اشیا،[[19]](#footnote-19) توسعه داده شده است . دیگر موتورهای استنتاج مبتنی بر الگوریتم Rete ، مانند RetePlus و Rete III که این الگوریتم با توجه به ضمینه کاربرد خود توسعه داده اند . عمومی ترین توسعه بر روی سیستم های آموزش استفاده شده که Rete/UL نام د ارد .

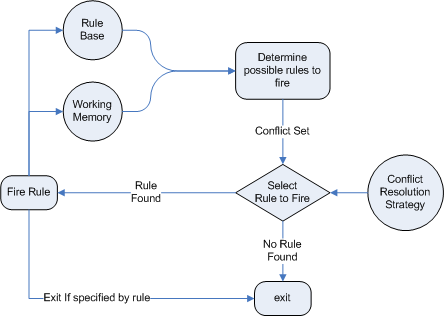
قوانین در حافظه تولید نگهداری[[20]](#footnote-20) می شوند و حقایق [[21]](#footnote-21) که موتور استنتاج آنهارا با قوانین تطابق میدهد در حافظه کاری[[22]](#footnote-22) نگهداشته می شوند . حقایق به حافظه کاری وارد می شوند و درآنجا ممکن است تغییر کنند. یک سیستم با تعداد زیادی از قوانین و حقایق . این امکان وجود دارد که چندین قانون با یک داده موجود در حافظه کاری مطابقت داشته باشد . در این صورت گفته می شود که این قوانین با هم تداخل دارند . دستورکار[[23]](#footnote-23) ترتیب اجرا این تداخل قوانین را با استفاده از استراتژی حل تداخلات [[24]](#footnote-24) مدیریت می کند.



شکل 1- نمایش سطح بالایی از موتور قوانین

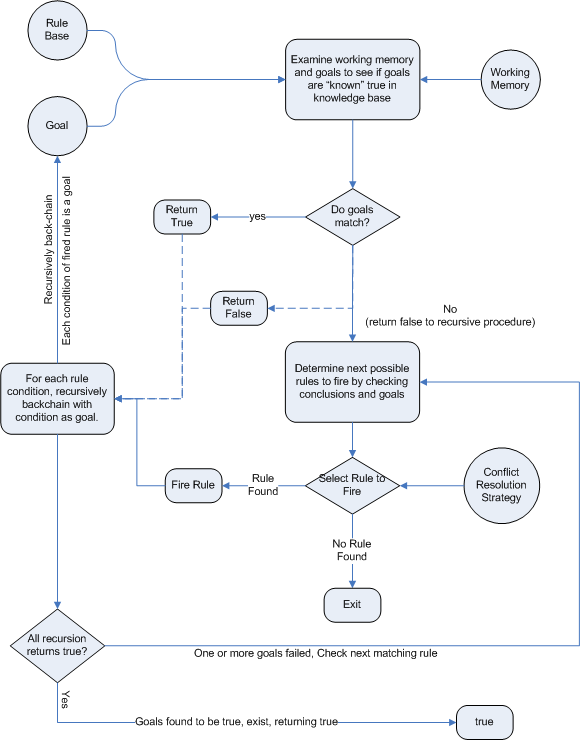
دو روش اجرایی برای سیستم قوانین وجود دارد : زنجیره ای رو به جلو[[25]](#footnote-25) و زنجیره ای رو به عقب [[26]](#footnote-26) سیستم هایی که ترکیبی از این دو روش استفاده میکنند سیستم های زنجیره­ای ترکیبی[[27]](#footnote-27) گفته می شوند . درک این دو روش عملیات مشخص میکند که چرا یک سیستم قوانین تولید متفاوت است و بهترین فرم بهره وری و استنتاج از قوانین را موجب می شود .

روش زنجیره­ای رو به جلو داده- محور [[28]](#footnote-28) است و در نتیجه به اصطلاح واکنشی است . با وارد شدن حقایق یا داده­ها به حافظه کاری ، یک یا چند قانون که مطابقت دارند و قسمت شرطی آنها مقدار درست برمی­گرداند بر اساس دستورکاری مشخص شده اجرا می­شوند . به اختصار، در این روش با یک داده شروع میشود که منتشر شده و با یک نتیجه گیری به اتمام می­رسد.



شکل 2- روش استنتاج زنجیره ای رو به جلو

روش زنجیره­ای رو به عقب روش به اصطلاح هدف- محور[[29]](#footnote-29) است . به این معنا که موتور استنتاج کارش را با یک نتیجه شروع می­کند که هدف رسیدن به آن است . اگر قانونی با مستقیما با این نتیجه ارتباط نداشته باشد . قوانینی جستجو می شود که بطور غیر مستقیم به هدف برسند . که اصطلاحا اهداف فرعی[[30]](#footnote-30) گفته می­شود . که کمک خواهد کرد تا بعضی از بخش های ناشناخته هدف اولیه ارضا شود . این فرآیند ادامه پیدامی­کند تا زمانیکه به هدف اولیه رسیده و یا هیچ قانون فرعی باقی نمانده باشد . زبان Prolog یک نمونه پیاده سازی از الگوریتم زنجیره رو به عقب است .



شکل 3- روش استنتاج زنجیره رو به عقب

## علل استفاده از موتور قوانین کسب و کار

سوالاتی که غالبا در استفاده از موتور قوانین ایجاد می شوند :

چه زمانی باید از موتور قوانین استفاده کرد ؟

استفاده از موتور قوانین چه مزیت هایی نسبت به کدهای دستی "if … then " دارد ؟

چرا باید از موتور قوانین به جای چارچوب های اسکریپتی مانند BeanShell استفاده کرد ؟

در زیر سعی می شود که به سوالات بالا پاسخ داده شود.

### مزیت های استفاده از موتور قوانین

**برنامه نویسی اعلانی** [[31]](#footnote-31)

موتورهای قوانین این امکان را فراهم می­کنند که گفته شود "چه کاری باید انجام شود" نه اینکه "چگونه انجام شود".

مهمترین مزیت این نکته این است که استفاده از قوانین، بیان کردن راه حل برای مسائل پیچیده و در نتیجه ارزیابی آن هارا تسهیل می­کند.

سیستم های قوانین کسب و کار قادر به حل مسائل بسیار پیچیده می­باشند . به همراه توضیحاتی برای اینکه چگونه یک راه حل حاصل شده است و هر تصمیم چگونه گرفته شده .

**جدا سازی منطق کسب و کار از داده**

داده های برنامه در اشیاء دامنه وجود دارند و منطق برنامه در قوانین . این شکستن ساختاری داده و منطق برنامه با توجه به اینکه از چه منظری نگاه کرده شود ممکن است مزیت و یا عیب باشد . نتیجه این است که اگر منطق برنامه بطور کامل در قوانین پیاده سازی شود قابلیت نگهداری و اعمال تغییرات آینده بر روی منطق برنامه بسیار ساده تر خواهد بود. و این مزیت زمانی بیشتر خودش را نشان می دهد که منطق کسب و کار بین­- دامنه­ای[[32]](#footnote-32) و یا چند دامنه­ای[[33]](#footnote-33) باشد . بجای اینکه منطق کسب و کار بین اشیاء یا کنترلر های چندین دامنه پخش شود می تواند درون فایل های قوانین جداگانه سازمان دهی شود .

**سرعت و مقیاس پذیری[[34]](#footnote-34)**

الگوریتم های انطباق که در موتورهای قوانین جهت استنتاج استفاده می شوند راه های بسیار کارایی را جهت پیداکردن انطباق در داده ها و قوانین ارائه می­کنند. این الگوریتم های در حالتیکه مجموعه داده ها بصورت جزیی تغییراتی می­کنند و موتورها قوانین انطباق های گذشته را به یاد دارد و با کارایی بیشتری قوانین مورد نظر پیدا می­کنند .

**مترکز سازی دانش**

با استفاده از قوانین، یک منبعی از دانش ایجاد می­شود که قابل اجرا است . این بدین معناست که یک نقطه متمرکز از سیاست های کسب و کار وجود خواهد داشت .و در حالت ایده­آل قوانین به قدری خوانا هستند که بتوانند به عنوان مستندات قابل استفاده باشند .

**ابزارهای یکپارچگی**

ابزارهای متفاوتی جهت توسعه و تغییر قوانین چه در محیط توسعه و چه در محیط اجرا و مبتنی بر وب موجود است . که بازخورد سریعی جهت اعتبار سنجی قوانین فراهم می­کند.

**سهولت شرح دهی و توصیفات اجرا**

سیستم های قوانین، مکانیزم هایی را فراهم می­کنند که مراحل اجرا و گزینش قوانین بطور کاملا کارا قابل پیگیری و ثبت باشند .

**قوانین قابل فهم**

با استفاده از ایجاد مدل های اشیاء و زبان خاص دامنه[[35]](#footnote-35) که با استفاده از آنها می­توان قوانینی نوشت که به زبان طبیعی بسیار نزدیک باشند. با استفاده از این امکانات متخصصین دامنه با کمترین اطلاعات تکنیکی می­توانند قوانین را ایجاد کنند و یا در صورت تغییر سیاست ها تغییر دهند .

### شرایطی که باید از موتورهای قوانین استفاده کرد .

کوتاه­ترین جواب به این سوال این است که :

*" زمانیکه هیچ راه حل برنامه نویسی سنتی رضایت بخشی برای حل مشکل نباشد"*

با این جواب کوتاه توضیحات بیشتری مورد نیاز خواهد بود . علت اینکه هیچ راه حلی سنتی وجود نداشته باشد یکی از موارد زیر است :

* **وقتی تعداد قوانین کسب و کار بیش از اندازه باشد.**

در شرایطی که ممکن است مسئله پیچیدگی نداشته باشد اما حجم قوانین آنقدر زیاد باشد که نتوان راه حل سنتی مناسبی برای آن ارائه داد .

* **مسئله ورای هیچ یک راه حل روشن الگوریتمیک باشد.**

در شرایطی که پیچیدگی زیاد است . و هیچ راه حل سنتی روشنی برای آن نباشد و یا اینکه مسئله بطور کامل درک نشده است .

* **شرایطی که منطق برنامه تغییر خواهد کرد .**

ممکن است خود منطق کسب و کار پیچیدگی نداشته باشد و ساده باشد اما منطق کسب و کار قرار است در طور عمر نرم­افزار تغییر کند . در بسیاری از سازمانها ، انتشار نسخه های جدید نرم افزار ها به تعداد کم و در فواصل زمانی زیاد انجام می­شود و قابلیت تغییر آسان قوانین کسب و کار بسیار مورد نیاز خواهد بود.

* **متخصصان کسب و کار آماده به کار هستند ولی فنی نیستند.**

متخصصان دامنه اغلب مالک ارزش دانش مربوط به قوانین کسب و کار فرآیندها هستند . آنها بطور عمومی غیرفنی هستند، اما می­توانند بسیار منطقی باشند . قوانین به آنها اجازه می­دهند که منطق کسب و کار را به روش خود اظهار کنند . خیلی از افراد در جایگاه غیر فنی، معمولا آموزش منطق رسمی ندیده اند ، و باید با کد کردن کسب و کار دانش در قوانین، این خلاء پر شود.

اگر قوانین تکنولوژی جدیدی برای تیم پروژه باشند، سر بار راه اندازی و آشنایی با این تکنولوژی باید در نظر گرفته شود .

بطور خاص در برنامه های کاربردی شی گرا مدرن، از موتور جستجو به عنوان بخشی از منطق کسب و کار درنظر گرفته می شود. مخصوصا بخشی که بسیار پیچیده است . این در حقیقت وارونگی مفهوم شی گرایی کپسوله کردن همه منطق درون اشیاء است.

اگر در پروژه بخش هایی باشندکه با عبارتهای شرطی تودرتو و پیچیده پیاده سازی شده باشد آن بخش دقیقا بخش هایی است که باید توسط قوانین پیاده سازی شوند . و اگر بخش هایی از کد وجود دارد که برنامه نویسان پی در پی در حال تغییر آنها هستند که ممکن است به دلیل اشتباهات یا تغییرات پیشنهاد شده توسط طراحان باشد آن قسمت دقیقا جایی است که قوانین و موتور قوانین میتواند نقش موثری داشته باشد و می توانید به آن فکر کنید.

قوانین می­توانند درون برنامه کاربردی جاسازی شوند [[36]](#footnote-36) و یا اینکه بعنوان سرویس استفاده شود . که البته برای بهره بردن از ویژگی حافظه دار[[37]](#footnote-37) موتور قوانین باید از ماژول موتور قوانین به عنوان بخشی از برنامه کاربردی استفاده کرد که در فصل های بعد ذکر خواهد شد.

برای سازمانی که قرار است از موتور قوانین استفاده شود . مهم است که تصمیم بگیرد که فرآیند به روز رسانی و یا اضافه کردن قوانین چه فرآیندی استفاده خواهد شد . گزینه های زیادی برای این کار وجود دارد، اما سازمان های متفاوت نیازمندیهای متفاوتی دارند . هرچند که تغییر کردن قوانین و سیاست های سازمان برای مالکین یا توسعه دهندگان پروژه قابل کنترل نیست .

## شرایطی که نباید از موتور قوانین استفاده کرد

باید توجه داشت که نباید فراموش کرد که یک موتورقوانین فقط بخشی از یک پروژه پیچیده و یا راه حل است. موتورهای قوانین قرارنیست که گردش کار یا فرآینداجرایی را به دست بگیرند و همچنین موتورهای گردش­کار و یا ابزارهای مدیریت فرآیند برای انجام قوانین کسب و کار طراحی نشده اند و باید از ابزار مناسب برای هر کار استفاده کرد.

همانطورکه موتورهای قوانین پویا هستند (پویا به این مفهوم که قوانین می توانند به عنوان داده ذخیره و مدیریت شوند)، اغلب به آنها به عنوان راه حلی برای مسئله توسعه نرم افزار دیده می شوند . اگر به این علت است که از موتور قوانین استفاده می شود، باید آگاه بود که موتورقوانین زمانی بهترین استفاده را دارند که از خاصیت اعلانی بودن قوانین استفاده کنید .به عنوان یک جایگزین، می توان از طراحی داده- محور یا موتورهای فرآیند اسکریپت بطوریکه اسکریپ ها در پایگاه داده مدیریت شده اند و قابلیت به روز رسانی را دارند .

## ویژگی های سیستم های مدیریت قوانین کسب وکار

با توجه به تعریف کوتاهی که توسط Tony Morgan بیان شده است. *" قانون کسب و کار یک عبارت فشرده درباره جنبه ای از کسب و کار است که می­تواند بگونه ای ساده و غیر مبهم بیان شود که توسط همه طرفین کسب وکار از جمله صاحبین ، تحلیلگر وطراحان و توسعه دهندگان کسب و کار قابل دسترسی باشد و این قوانین در یک نظام تعبیه شده اند تا با هم در ارتباط باشند"*.

عبارت *x= y-z* یک قانون نیست . این یک عبارت انتساب است و یا به عبارت دیگر یک رویه[[38]](#footnote-38) محاسبه x است. یک قانون در سیستم مدیریت قوانین کسب و کار غیر – رویه ای[[39]](#footnote-39) است . آنها مشخص می­کنند که چه چیزی درست است نه اینکه چگونه محاسبه شوند . همچنین برقراری اتباط بین قوانین را استنتاج[[40]](#footnote-40) می گویند . یک سیستم مدیریت قوانین علاوه بر پشتیبانی از حقایق و قوانین باید از استنتاج خودکار پشتیبانی کند .

در این بخش مروری بر ویژگی های تکنیکی سیستم های مدیریت قوانین کسب و کار خواهیم داشت. این ویژگی ها شامل موارد زیر می شود :

* معماری
* یکپارچه سازی با برنامه کاربردی تجاری
* بازنمایی دانش
* استراتژی های استنتاج
* تبدیل کردن قوانین به مجموعه قوانین مستقل و قابل اتصال
* وضعیت ویژگی هایی که در سیستم های مدیریت قوانین کنونی دیده می شوند .
* ایجاد برنامه های کاربردی هوشمند که بتواند با استفاده از گفتگو منطقی و قابل فهم با کاربر ارتباط برقرار کند .
* دادن امکان ایجاد و تغییر و مدیریت قوانین کسب و کار به کاربران و طراحان سیستم .
* مدیریت نسخه[[41]](#footnote-41) و مخازن[[42]](#footnote-42) قوانین.

### کامپوننت ها و ویژگی های فنی سیستم مدیریت قوانین کسب و کار (BRMS[[43]](#footnote-43))

سیستم های مدیریت قوانین زیرمجموعه سیستم های خبره­ای هستند که از سال 1980 تولید شده اند . سیستم های قانون محور یا خبره که گاهی اوقات سیستم های دانش – محور نیز گفته می شوند . سیستم های کامپیوتری هستند که می توانندپیشنهاد دهند و یا تصمیم گیری کنند. دونوع از این سیستم ها وجود دارد : سیستم هایی که تصمیم گیری میکنندکه بطور کلی کنترلر ها و برنامه های کاربردی مانند سیستم­های مالی ، سیستم های تجاری و سیستم هایی که به عنوان سیستم های پشتیبانی از تصمیم گیری عمل میکنند،آنها پیشنهاد می دهند اما تصمیم نمیگیرند . این تعریف نشان دهنده بخش از کارهایی است که سیستم های خبره انجام میدهند . مهم تر اینکه، سیستم های قانون- محور با توجه اینکه چه معماری دارند تعریف می شوند . مهمترین ویژگی معماری اینگونه سیستم ها این است که دانش مربوط به مسئله جدا از کدی که دانش را بر مسئله در دست اعمال میکند، نگهداری میشود.

مخزن دانش در یک سیستم مدیریت قوانین بعنوان **پایگاه قوانین**[[44]](#footnote-44) یا **پایگاه دانش[[45]](#footnote-45)** رجوع می شود و مکانیزمی که دانش را بر داده های ارائه شده اعمال می کند **موتور قوانین** یا **موتور استنتاج[[46]](#footnote-46)** گفته می شود.

شکل 4- معماری سطح بالا سیستم مدیریت قوانین کسب و کار

پایگاه دانش

موتور استنتاج

اکنون بطور گسترده پذیرفته شده است که سیستم مدیریت قوانین از چهار بخش اصلی تقسیم شده است . بخش اول، محیط اساسی از سمبل و تغییر مقادیر که همه سیستم های کامپیوتری به اشتراک می گذارند که میتواند به عنوان زبان برنامه نویسی در نظر گرفته شود . مانند ویرایشگرها و کامپایلرها که بخش خاکستری شکل بالا این بخش را نشان می دهد .

بخش دوم، ساختار پایگاه دانش است که شامل روش های بازنمایی و دسترسی به دانش نیز می شود و نهایتا باید تکنیک هایی برای اعمال دانش برای مسئله در دست وجود داشته باشد .

بخش سوم، موتور استنتاج است که قوانین را به هم زنجیر میکند تا به نتیجه معتبر برسد .

بخش چهارم،مخزنی است که در آن قوانین نگهداری می شوند و قابل ویرایش و مدیریت نسخه است .

پایگاه دانش و موتور استنتاج از یکدیگر جدا در نظر گرفته شده اند تا نگهداری را تسهیل بخشند . علاوه بر این در بیشتر موارد قوانین و سیاست ها به مرور زمان تغییر میکنند در حالیکه موتور استنتاج قرار نیست بازنویسی شوند .

پایگاه دانش معمولا شامل انواع مختلف دانش می شود : دانش مربوط به اشیاء،روندها و ارتباطات . دانش در رابطه با اشیاء معمولا در فرم مدل اشیاء ، شمای XML ، مدل داده و شبکه معنایی ذخیره می شود . دانش روند ممکن است بصورت متدهای جاوا یا ماکروهای اکسل بازنمایی شوند . بعضی از روندهای کسب و کار نیز توسط قوانین بازنمایی می شوند.

### قوانین

دانش مربوط به ارتباطات معمولا بصورت قوانین به فرم "IF A THEN B" نگهداری می شوند . برخلاف عبارت های if/then که در زبان های رسمی مانند جاوا پیدا می شوند . زبان های قوانین عموما **اعلانی**[[47]](#footnote-47) با **غیر–رویه ای** [[48]](#footnote-48) هستند به این معنا که ترتیبی که قوانین نوشته می شوند در نتیجه نهایی تاثیری ندارد . این قوانین بر روی دانش موجودیت ها و اشیاء کار میکنند .همانجور که گفته شد . راه دیگر بازنمایی دانش رویه ها هستند. مانند رویه هایی که در زبان های مرسوم وجود دارند. راه های دیگری نیز برای بازنمایی دانش وجود دارد اما قوانین ، رویه ها و اشیا مهمترین آنها هستند.

### قالب های قوانین[[49]](#footnote-49)

قالب های قوانین الگوهای طراحی برای قوانین هستند. در بسیاری از شرایط یک قانون ممکن است برای داده های مختلف قابل اعمال باشند . در اینچنین مواردی، قالب های قوانین قابلیت ایجاد قوانین با یک جایگاه خالی که قرار است بعدا پر شود را میدهد . یک قالب قانون کسب و کار یک قانون کسب و کار تعریف شده نسبی را که شامل جانما[[50]](#footnote-50)ها برای داده هایی است که بعدا پر می شود . در اینصورت با پرکردن جاهای خالی با مقادیر جدید قوانین جدید تولید خواهد شد .

### چک کردن نحوی قوانین[[51]](#footnote-51)

یک موتور قوانین خوب امکاناتی را برای بررسی بی وقفه نحو قوانین، همزمان با نوشتن قوانین فراهم میکند. با زبان های قوانین ساخت یافته ، بسیار مفید است که ابزار مناسبی جهت چک کردن صحت نحوی قوانین ، نمایش کلمات کلیدی و متغییرها با رنگ های متفاوت وجود داشته باشد .

### رویه ها و الگوریتم ها

بعضی دانش ها از نوع رویه ای هستند . برای مثال قانون محاسبه مالیات . بازنمایی دانش زمانیکه قرار است در رویه ها نگهداری شوند، می تواند بسیار سنگین باشد. یک سیستم مدیریت قوانین خوب به مجموعه قوانین امکان استفاده از رویه های را میدهد. بطوریکه در اجرای قوانین قابلیت اجرا و برگرداندن قوانین را داشته باشد.

### جریان قوانین[[52]](#footnote-52)

مکانیزم جریان قوانین باید در سیسستم مدیریت قوانین وجود داشته باشدکه به طراحان این اجازه را میدهد که ماژول ها وقوانین با ترتیب یا اولویت مشخصی اجرا شوند . این ویژ/ی برای یک سیستم مدیریت قوانین خوب الزامی است .

### جداول و درخت تصمیم گیری[[53]](#footnote-53)

در یعضی از محصولات، بازنمایی جایگزینی برای بازنمایی قوانین بصورت if/then وجود دارد .و این دو نوع بازنمایی یعنی درخت تصمیم گیری و جدول تصمیم گیری در نظر گرفته شده است . درختان تصمیم گیری قوانین را بصورت ساختار درختی نمایش می دهد .این نوع بازنمایی برای عیب یابی و ارتباط بین کاربران و برنامه نویسان و طراحان کسب و کار مفید است. جداول تصمیم گیری دانش مشابهی را بصورت ساختار جدولی نمایش می دهند .

مشکل اصلی جداول تصمیم گیری این است که معمولا در حالتی که قوانین زیاد باشند بصورت غیر مدیریت شده ای رشد میکند . در این نوع بازنمایی هر ردیف جدول یک قانون را معرفی می کنند که خوانایی کمی دارد.

مهمترین مزیت جداول تصمیم گیری برای حالتی است که این قوانین از قبل در سازمان بصورت جدولی وجود داشته باشد . مانند استفاده از جدول قیمت یا جدول نرخ سود بعنوان بخشی از قوانین کسب و کار که در این حالت بهتر است از این داده های جدولی بعنوان داده های قوانین استفاده شود .

### استنتاج

یک موتور استنتاج یک یا چند ابزار را برای اعمال دانش بر روی داده پیشنهاد میکند . مهمترین استراتژی ها **زنجیره رو جلو** و **زنجیره رو به عقب** نام دارند .

روش زنجیره­ای رو به جلو داده- محور [[54]](#footnote-54) است و در نتیجه به اصطلاح واکنشی است . با وارد شدن حقایق یا داده­ها به حافظه کاری ، یک یا چند قانون که مطابقت دارند و قسمت شرطی آنها مقدار درست برمی­گرداند بر اساس دستورکاری مشخص شده اجرا می­شوند . به اختصار، در این روش با یک داده شروع میشود که منتشر شده و با یک نتیجه گیری به اتمام می­رسد.

روش زنجیره­ای رو به عقب روش به اصطلاح هدف- محور[[55]](#footnote-55) است . به این معنا که موتور استنتاج کارش را با یک نتیجه شروع می­کند که هدف رسیدن به آن است . اگر قانونی با مستقیما با این نتیجه ارتباط نداشته باشد . قوانینی جستجو می شود که بطور غیر مستقیم به هدف برسند . که اصطلاحا اهداف فرعی[[56]](#footnote-56) گفته می­شود . که کمک خواهد کرد تا بعضی از بخش های ناشناخته هدف اولیه ارضا شود . این فرآیند ادامه پیدامی­کند تا زمانیکه به هدف اولیه رسیده و یا هیچ قانون فرعی باقی نمانده باشد . زبان Prolog یک نمونه پیاده سازی از الگوریتم زنجیره رو به عقب است .

### عدم قطعیت[[57]](#footnote-57) و شرح دهی[[58]](#footnote-58)

دو ویژگی مهمی که سیستم های قانون محور را از دیگر سیستم های کامپیوتری متمایز میکند، این است که آنها اغلب میتوانند :

* شرح چگونگی نتیجه گیری را فراهم کنند .
* ترکیب استدلال کیفی یا قضاوت و مدیریت عدم قطعیت.

اگر این دو ویژگی با هم وجود داشته باشند، سیستم های قانون محور ممکن است بیش یک نتیجه همراه امتیاز دهی هر نتیجه پیشنهاد دهند . که البته در صورت نیاز این دو ویژگی اعمال آنها در سیستم باعث هزینه و پیچیدگی بیشتر سیستم می شود .

تسهیلات تعبیه شده شرح دهی ابزار مناسبی جهت رفع باگ و بررسی مسیر تصمیم گیری ارائه می کنند اما به ندرت برای کاربر نهایی قابل توجه است . امکانات مفید برای شرح دهی در استنتاج برای کاربران باید دست ساز باشند.

چندین تکنیک برای مدیریت عدم قطعیت که بسیار رایج است وجود دارد:

• استدلال صریح با استفاده از برچسب کلامی برای شرایط نامشخص

• امکانات What-if

• سیستم های تعمیر و نگهداری حقیقت

• عامل اطمینان

• احتمال Bayesian

• مجموعه های فازی

استدلال در مورد عدم قطعیت به پیچیدگی یک سیستم می افزاید و کسب دانش در ارتباط با آن است، اما اجازه مقابله با مشکلات پیچیده تر را به ما میدهد.

فصل دوم

بررسی سیستم های مدیریت قوانین کسب و کار موجود

# بررسی سیستم های مدیریت قوانین کسب و کار موجود

## مقدمه

تولیدات زیادی وجود دارند که به کاربران اجازه می دهندتا سیستم های مبتنی بر قاونون را توسعه دهند. البته ممکن است همه آنها به عنوان سیستم مدیریت قوانین کسب و کار دسته بندی نشوند . در این بخش فاکتورهایی در نظر گرفته می شود که تعداد زیادی از آنها را از لیست کاندید حذف شوند . شروطی که اعمال شده اند که لیست باند سیستم های کاندید کوتاه شوند به اختصار در زیر آمده است.

* باید اجازه دهند تا تحلیلگران کسب و کار قوانین را ایجاد کنند و تغییر دهند.
* استفاده از یک مخزن دانش با ویژگی های کامل.
* مبتنی بر الگوریتم rete باشد.
* پشتیبانی از روش زنجیره عقب.
* باید امکان استفاده از موتور قوانین به عنوان یک جزء و یا سرویس در برنامه های کاربردی بزرگتر وجود داشته باشد.
* باید اجازه دهد تا برنامه های کاربردی در معماری سرویس گرا مستقر می شود.
* تمرکز برمسائل مدیریت قوانین کسب و کار .
* مدارک و شواهد از برنامه های کاربردی تجاری موفق استفاده از آن را فراهم کند.
* منبع باز بودن سیستم مدیریت قوانین.

ملاحضات ذکر شده باعث حذف زیادی از سیستم های کاندید استفاده می شود . در ادامه این فصل لیست کامل سیستم های کاندید معرفی می شوند و مورد مطالعه و بررسی قرار می گیرند .

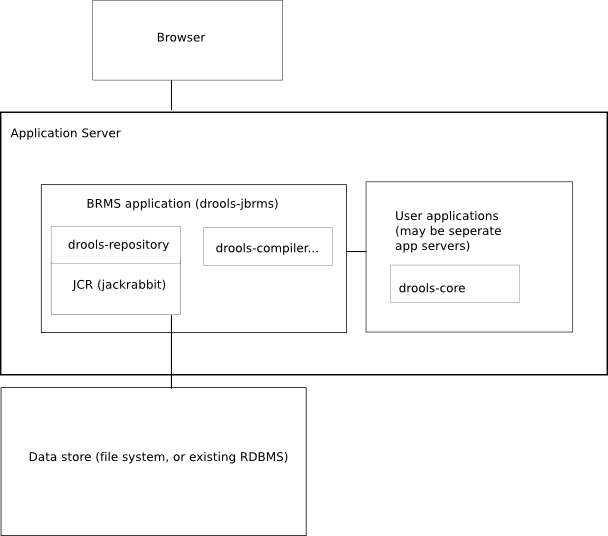
## معرفی سیستم های مدیریت قوانین موجود

### Drools

Drools یک سیستم مدیریت قوانین کسب وکار توسعه یافته است که بر اساس الگوریتم Charles Forgy's Rete پیاده سازی شده است و برای [[59]](#footnote-59)JVM مناسب است .Drools امکان برنامه نویسی را تامین میکند و برای تطبیق مسئله دامنه کسب و کار با زبان های خاص، ابزارهای ویرایش گرافیکی، ابزارهای مبتنی بر وب و ابزارهای بهره وری طراحان فراهم کرده است که بسیار انعطاف پذیر است . Drools یک مجوز کسب و کار متن باز دارد که اجازه دانلود، استفاده، تعبیه و توزیع را بطور رایگان را ارائه میکند .به کاربران اجازه میدهد که بیان کنند چه انجام شود و به چگونگی انجام آن کاری ندارد . بصورتی که هم برای کاربران و هم برای طراحان قابل فهم باشد .زبان های خاص دامنه [[60]](#footnote-60) به طراحان اجازه میدهد سمنتیک زیان طبیعی را به عنوان قاعده بنویسند. به این ترتیب فاصله بین توسعه دهندگان و طراحان غیر فنی را کاهش میدهد .در آخرین نسخه مجموعه ابزار Drools محیطی گرافیکی به نام Kie-Workbench [[61]](#footnote-61) توسعه داده شده که محیطی برای استفاده از کلیه امکانات این ابزار فراهم میکند .

گروه توسعه Drools افزونه هایی[[62]](#footnote-62) را جهت تسهیل توسعه دهندگان در محیط توسعه Eclipse توسعه داده اند که نوشته قوانین را در محیط توسعه و اعتبار سنجی وتست آنها آسان تر کرده است .

معماری سیستم Drools و چگونگی ارتباط با برنامه کاربردی در شکل زیر نمایش داده شده است :



شکل 5- معماری سیستم مدیریت قوانین Drools

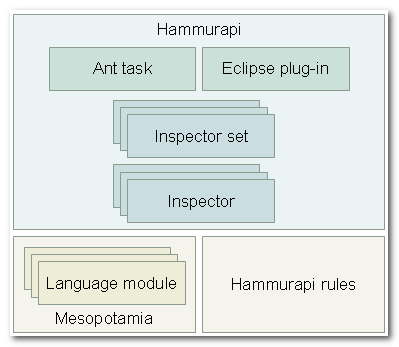
### Jess

Jess یک موتور قوانین و یک محیط اسکریپت است که توسط Emest FriedMan در آزمایشگاه های Sandia National در Livermore به زبان جاوا نوشته شده است . با استفاده از Jess می توان یک نرم افزار جاوا توسعه داد که با استفاده از اطلاعاتی که در فرم های قوانین وجود دارد،توانایی استدلال داشته باشد. Jess کوچک، سبک و یکی از سریعترین موتور قواعد موجود است .زبان اسکریپتی و قدرتمند آن، دسترسی به همه API های جاوا را ممکن می سازد. شامل یک محیط توسعه یافته با تمام ویژگی های سکوی Eclips است . از یک نسخه پیشرفته الگوریتم Rete برای قوانین فرآیند استفاده میکند. Rete یک مکانیزم کارا برای حل پیچیده تطبیق است. همچنین از متدهای زنجیره رو به عقب و زنجیره رو به جلو پشتیبانی می کند . در رابطه با اشیاء جاوا، میتواند مستقیما بطور دستی عمل کرده و استدلال کند . یک محیط اسکریپت جاوا دارد که می توان در آن اشیاء جاوایی ایجاد کرد . متدهای جاوا را فراخوانی کرد و واسط های جاوا را بدون کامپایل کرهای جاوا پیاده سازی کرد. برای استفاده های تجاری تایید شده استو برای استفاده های آموزشی بدون هزینه در دسترس است .

### Hammurapi Rules

Hammurapi Rules یک موتور قواعد برای توسعه دهندگان جاوا است . قوانین در آن با زبان جاوا طراحی شده اند. از متد زنجیره رو به جلو وروبه عقب برای استنتاج پشتیبانی می کند . طراحانی که با زبان جاوا آشنایی دارند لازم نیست برای نوشتن قوانین زبان خاص دیگری یاد بگیرند . . همه اطلاعات برای توسعه جاوا با یکریگر تلفیق می شوند و می توانند برای توسعه قوانین به کار روند. به ابزارها و فرآیند های خاصی نیازی نیست . قوانین منطقی نیز مانند دیگر کلاس های جاوا به JVM و منابع سازمان دسترسی دارند . قوانین بوسیله کامپایلرهای جاوا بر حسب bytecode کامپایل می شوند و JVM بهینه سازی قوانین را در زمان اجرا تامین میکند . اجرای موازی قوانین نیز امکان پذیر است .بطور کلی ویژگی های زیر را دارد :

* پشتیبانی از زنجیره رو به جلو از طریق واسط های JSR-94
* پشتیبانی از زنجیره رو به عقب در مواقعی که از زنجیره رو به جلو کارایی بیشتری دارد .
* استفاده از کد جاوا ساده برای نوشتن قوانین .
* قوانین با استفاده از ساختار XML پیکربندی و اسمبل می شوند .
* از واسط چند نخی پشتیبانی می کند .



شکل 6 - معماری کامپوننت های Hammurapi Rules

### InfoSapient

InfoSapient برنامه ای برای بیان آسان اجرا و نگهداری از قوانین کسب و کار در یک شرکت است و کاملا بر اساس استاندارد جاوا نوشته شده است . یک موتور قوانین است که به اعضای انجمن کسب و کار اجازه میدهد که قوانین کسب و کاری که روزانه استفاده می کنند، ثبت کرده و نگهداری کنند. بسیاری از متغییر ها بطور همزمان برای سنجیدن ریسک ها و فرصت ها برای رسیدن به بهترین حالت در نظر گرفته می شوند . InfoSapient موقعیت های مبهمی را که در فرآیندهای کسب و کار ایجاد می­شوند به بهترین حالت راه اندازی می کند . بدون نیاز به مهارت و تجربه ی نیروی انسانی تصمیمات پیچیده و به موقع گرفته می شود. با استفاده از apache Tomcat و IBM WebSphere با موفقیت آزموده شده است .

### JLisa

JLisa یک بستر قوی برای قوانین کسب و کار قابل دسترس به جاوا بوده و با JSR-94 سازگار است . از Clips قدرتمندتر است زیرا مزایاط بسط یافته از ویژگی های عمومی Lisp را دارد . این ویژگی ها برای نرم افزار توسعه یافته multi-paradign ضروری هستند . Jlisa بر اسا ABL1 و Lisa است و قابلیت های آن عبارتند از :

* اجرای قوانین Clips-like روی اشیاء جاوا
* ساخت سیستم های Java agent که قابلیت استدلالی دارد .
* ساخت سیستم های جاوا با مشارکت اشیاء جاوا
* ساخت GPS در جاوا
* اجرای ارزیابی قانون کسب و کار
* ایجاد و اجرای درخت تصمیم برای اشیاء جاوا
* اجرای جستجوها و پرسش و پاسخ های منطقی روی اشیاء جاوا
* ایجاد smart Wizard و غیره...

### Mandarax

Mandarax یک موتور قوانین است که کاملا بر اساس جاوا پیاده سازی شده است . انواع قوانین بر اساس پایگاه داده­ها، reflection ، EJB و استانداردهای XML پشتیبانی می شوند . با استفاده از زنجیره رو به عقب یک موتور استنتاج J2EE را تامین میکند .

### OpenLexicon

OpenLexicon یک ابزار مدیریت فرآیند و قوانین کسب و کار است که برنامه های کاربردی را برای تراکنش برنامه های مبتنی بر فرآیند توسعه می دهد . برای مدیریت و آزمئن قوانین کسب وکار، شامل یک انبار فراداده، یک موتور قوانین کسب کار و یک واسط کاربری جامع مبتنی بر وب است .

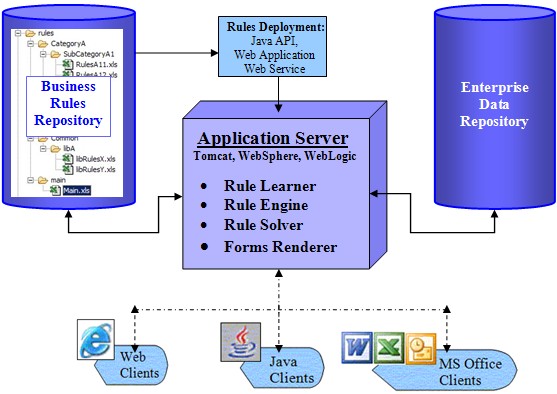
برای هماهنگ سازی تعاملات پیچیده در قوانین کس بو کار و اشیاء کسب و کار دارای ابزارهای مدیریت فرآیند ها می باشد. شهرت Openlexicon یه علت تامین راه کارهایی با کارایی بالاست که در مجموعه ای از برنامه های کاربردی در سطح سازمانی استفاده می شود. دارای دو بخش اصلی است که عباتند از انبار فراداده و موتور قوانین کسب وکار .

بخش مهم openLixicon به عنوان نرم افزار متن باز منتشر شده است . علاوه بر آن یک ویزارد دارد که یک ازار برای ساخت قوانین و موردهای کاربری در کسب و کار است . ویزارد برای طراحان و تحلیلگرانی طراحی شده است که از مهارت تکنیکی بالایی برخوردار نیستند .

### OpenRules

openRules یک متدولوژی ، ابزارهای متن باز و سرویس هایی را ارائه کرده است که تحلیلگران کسب و کار را قادر می سازد که با طراحان نرم افزار برای ایجاد و پشتیبانی از برنامه های کاربردی مبتنی بر قاعده در کسب و کارهای پیچیده و ارائه منطقی همکاری کنند . OpenRules یک سیستم مدیریت قوانین کسب و کار کامل است . بخش های ساده و قدرتمند متن باز را برای برنامه های کاربردی توسعه یافته با استفاده از قوانین تامین می کند . با استفاده از ابزارهایی مثل اکسل ، Google spreadsheets و Eclips IDE می توان تصمیمات پیچیده ای را برای پشتیبانی از سیستم های در محدوده یک سازمان ایجاد کرد .

openRules تنها برای استفاده از آموزشی و بررسی لیسانس محدود دارد.



شکل 7 - معماری سیستم مدیریت قوانین OpenRules

### Jamacho (morendo)

Jamocha یک موتور قوانین متن باز است که هدف آن تامین یک موتور قواعد با کیفیت بالا و یک محیط system shell خبره است. به علاوه ابزارهای توسعه یافته ی مورد نیاز و بهترین متدولوژی عملی را برای برنامه های کاربردی مبیتنی بر قوانین تامین میکند .

### JRuleEngine

‏ JRuleEngine ‏یک موتور قواعد جاوا است و ‏بر اساس خصوصیات Request94 ‏ نسخه ی 1.1 ‏است. قوانین می تواند به وسیله ی یک فایل xml ‏ یا JRuleEngine APIs ‏ بارگذاری شود ودر یک پایگاه داده ذخیره شوند. این توزیع شامل یک کتابخانه است که می تواند در هر نوع برنامه ی کاربردی جاوا ادغام شود،در نتیجه می تواند در هر نوع برنامه ی کاربردی ( بر اساس وب یا غیره) استفاده شوند.

JRuleEngine در وظیفه مندی و کارایی ضعیفتر از دیگر موتورهای قاعده عمل می کند. در مورد دسترسی به راهنمای کاربری این موتور می توان وضعیت خیلی ضعیف را در نظر گرفت. نقطه قوت این ابزار متن باز بودن و استفاده از ابزارهای مبتنی بر جاوا می باشد.

## مقایسه موتور قوانین کسب و کار

موتور های قوانین نام برده در گزارشی با عنوان "گزارش مطالعه و اولویت بندی ابزارهای BPMS " که توسط مرکز تحقیقات صنایع انفورماتیک انجام شده است استخراج شده است .

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | وظیفه­مندی ابزارهای مورد استفاده | کارایی ابزارهای مورد استفاده | سازگری ابزارهای مورد استفاده با یکدیگر | **متن باز بودن ابزار های مورد استفاده** | **کاربر پسند بودن ابزار مورد استفاده** | **دسترسی به راهنمای کاربری ابزار مورد استفاده** | **استفاد از ابزارهای مبتنی بر جاوا** | **پشتیبانی از استانداردهای مرتبط** | **پشتیبانی از سرویس های مختلف در گذرگاه سرویس** | **پشتیبانی از زبان های هماهنگ سازی سرویس ها** | **به روز بودن ابزار مورد استفاده** | **میزان استفاده سایرین از ابزار مورد نظر** | **موجود بودن تیم پشتیبانی قوی برای ابزار مورد نظر** | **دورنمای توسعه ابزار مورد استفاده** | **لیسانس نرم افزاری برای ابزار مورد استفاده** | **مجموع امتیازات** |
| **Drools** | **5** | **5** | **3** | **5** | **4** | **4** | **5** | **5** | **0** | **0** | **4** | **4** | **4** | **4** | **4** | **56** |
| **Jamocha** | **5** | **5** | **3** | **5** | **3** | **2** | **5** | **5** | **0** | **0** | **4** | **3** | **3** | **3** | **3** | **49** |
| **Jess** | **5** | **5** | **3** | **5** | **3** | **2** | **5** | **5** | **0** | **0** | **4** | **3** | **3** | **3** | **3** | **49** |
| **Hammurapi** | **3** | **3** | **2** | **5** | **3** | **3** | **5** | **3** | **0** | **0** | **4** | **2** | **3** | **3** | **3** | **42** |
| **InfoSapient** | **2** | **3** | **2** | **5** | **2** | **3** | **5** | **1** | **0** | **0** | **1** | **1** | **1** | **1** | **3** | **30** |
| **JLisa** | **2** | **3** | **2** | **5** | **3** | **3** | **5** | **3** | **0** | **0** | **3** | **3** | **3** | **2** | **3** | **40** |
| **Mandarax** | **2** | **3** | **2** | **5** | **2** | **2** | **5** | **1** | **0** | **0** | **3** | **2** | **2** | **2** | **3** | **34** |
| **OpenLexicon** | **2** | **2** | **1** | **5** | **2** | **2** | **5** | **1** | **0** | **0** | **2** | **2** | **2** | **2** | **3** | **31** |
| **Open Rules** | **2** | **2** | **1** | **5** | **2** | **3** | **5** | **1** | **0** | **0** | **5** | **2** | **2** | **2** | **3** | **35** |
| **Jrule Engine** | **1** | **1** | **1** | **5** | **1** | **1** | **5** | **1** | **0** | **0** | **1** | **1** | **1** | **1** | **3** | **23** |

بر اساس مقایسات انجام شده ، استفاده از سیستم مدیریت قوانین Drools پیشنهاد می شود که ابزاری متن باز و رایگان به زبان جاوا است که قابلیت مجتمع سازی بصورت تعبیه شده و مبتنی بر سرویس را دارد . همچنین ابزارهای تحت وب مناسبی برای ایجاد و دستکاری قوانین فراهم میکند .

در ادامه چگونگی استفاده و مجتمع سازی این سیستم با برنامه کاربردی تجاری مورد نظر شرح داده خواهد شد .

فصل سوم

معرفی موتور قوانین Drools

# فصل سوم : معرفی موتور قوانین Drools

## مقدمه

Drools یک سامانه قوانین کسب و کار است . که یک موتور قوانین کسب و کار ، یک برنامه کاربردی تحت وب جهت مدیریت و ویرایش قوانین و پلاگین هایی برای Eclipse برای توسعه دهندگان فراهم میکند. بطور کلی سامانه Drools شامل پروژه های زیر میباشد .

 Drools Workbench http://drools.org/headerFooter/guvnor_icon_32x.png

این پروژه محیطی تحت وب را برای ایجاد و ویرایش قوانین و تعریف فرآیند های فراهم میکند

 Drools Expert http://drools.org/headerFooter/expert_icon_32x.png

این پروژه بعوان هسته اصلی سامانه Drools مورد استفاده قرار می گیرد و در حقیقت موتور قوانین کسب و کار است . که در این پروژه از این ماژول استفاده شده است .

 Drools Fusion http://drools.org/headerFooter/fusion_icon_32x.png

ابزاری برای توسعه سیستم های مبتنی بر رویداد است . مدیریت رویدادهای پیشرفته را پشتیبانی می کند.

[jBPM](http://www.jboss.org/jbpm)  http://drools.org/headerFooter/flow_icon_32x.png

ابزاری جهت مدیریت فرآیندهای کسب و کار فرآهم میکند. و استاندارد های JBPM استفاده شده است .

[OptaPlanner](http://www.optaplanner.org/)  http://drools.org/headerFooter/planner_icon_32px.png

ابزاری جهت برنامه ریزی بهینه در پروژه هایی که نیاز به این گونه کسب و کار دارند فراهم می کند .

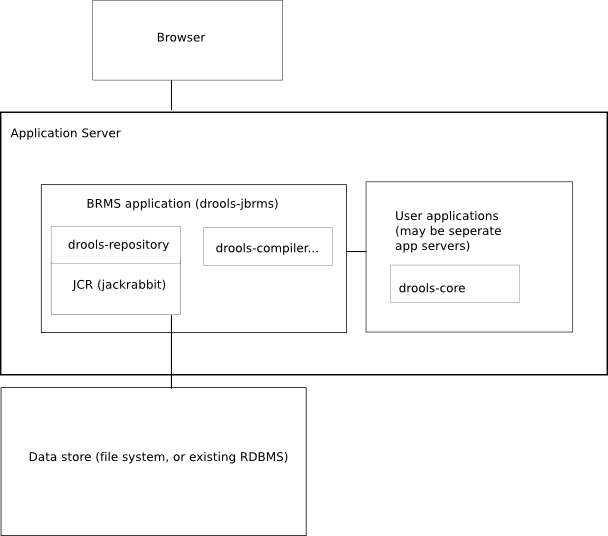
در پروژه جاری از موتور قوانین کسب و کار Drools یعنی پروژه Drools Expert استفاده شده است که در حقیقت نوعی از سیستم های خبره محسوب می شود . که در ادامه توضیحاتی ارائه می شود .

### موتور قوانین Drools - Drools Exper

Drools expert یک موتور مدیریت قوانین کسب وکار توسعه یافته است که بر اساس الگوریتم Charles Forgy's Rete پیاده سازی شده است و برای [[63]](#footnote-63)JVM مناسب است .Drools امکان برنامه نویسی را تامین میکند و برای تطبیق مسئله دامنه کسب و کار با زبان های خاص، ابزارهای ویرایش گرافیکی، ابزارهای مبتنی بر وب و ابزارهای بهره وری طراحان فراهم کرده است که بسیار انعطاف پذیر است . Drools یک مجوز کسب و کار متن باز دارد که اجازه دانلود، استفاده، تعبیه و توزیع را بطور رایگان را ارائه میکند .به کاربران اجازه میدهد که بیان کنند چه انجام شود و به چگونگی انجام آن کاری ندارد . بصورتی که هم برای کاربران و هم برای طراحان قابل فهم باشد .زبان های خاص دامنه [[64]](#footnote-64) به طراحان اجازه میدهد سمنتیک زیان طبیعی را به عنوان قاعده بنویسند. به این ترتیب فاصله بین توسعه دهندگان و طراحان غیر فنی را کاهش میدهد .در آخرین نسخه مجموعه ابزار Drools محیطی گرافیکی به نام Kie-Workbench [[65]](#footnote-65) توسعه داده شده که محیطی برای استفاده از کلیه امکانات این ابزار فراهم میکند .

گروه توسعه Drools افزونه هایی[[66]](#footnote-66) را جهت تسهیل توسعه دهندگان در محیط توسعه Eclipse توسعه داده اند که نوشته قوانین را در محیط توسعه و اعتبار سنجی وتست آنها آسان تر کرده است .

معماری سیستم Drools و چگونگی ارتباط با برنامه کاربردی در شکل زیر نمایش داده شده است :



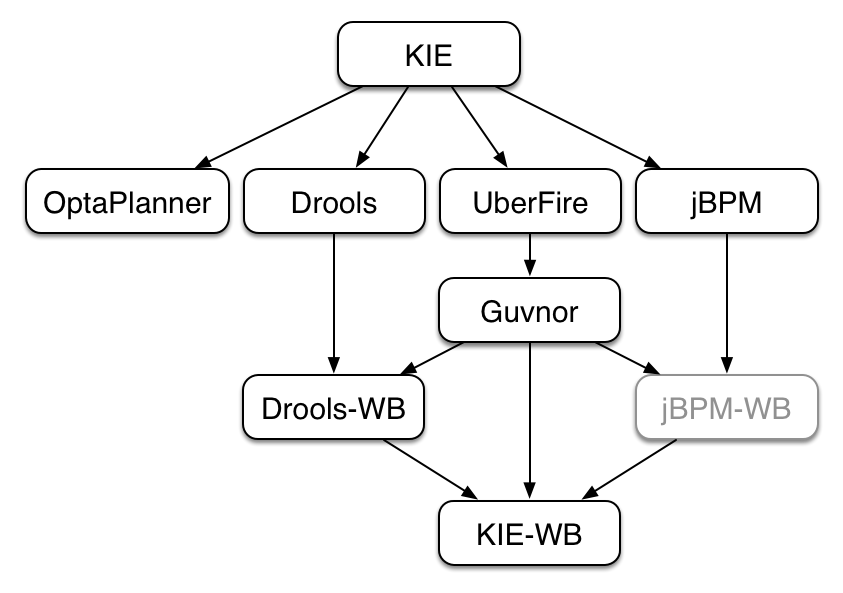
شکل 5- معماری سیستم مدیریت قوانین Drools

## معرفی KIE

### آناتومی پروژه

این پروژه یک پروژه متن باز تحقیقاتی است که با هدف ایجاد یک راه حل یکپارچه سازی دانش برای Drools و JBPM شکل گرفته است که در ابتدا نام droolsjbpm را انتخاب کردند . ولی سپس نام خود را به KIE به معنای "دانش همه چیز است"[[67]](#footnote-67) تغییر دادند .

در حال حاضر KIE شامل زیر پروژه های زیر است .



شکل ‏3‑1 ) آناتومی پروژه KIE

### چرخه های حیات دانش در KIE

جنبه های متفاوت یا چرخه حیات در KIE چه در Drools ویا jBPMS میتواند بطور عمومی به بخش های زیر شکسته شود .

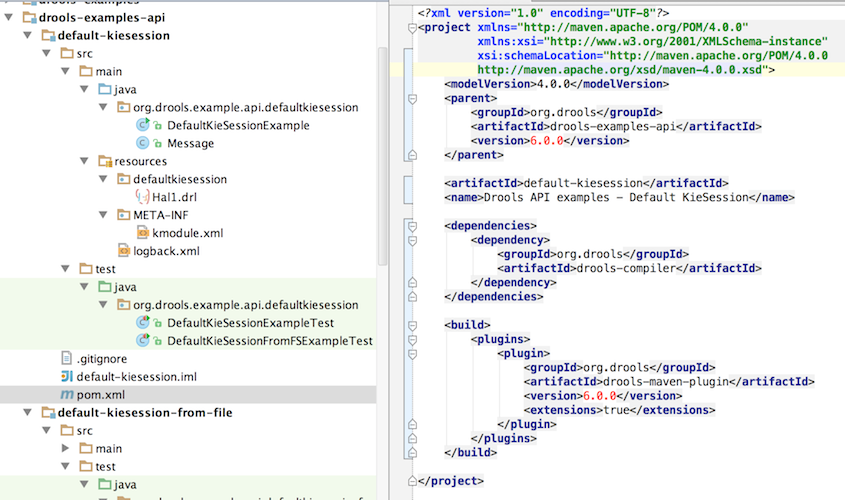
* **نوشتن دانش**
* نوشتن دانش با استفاده از واسط های UI، مانند: DRL، BPMN2، جدول تصمیم گیری و مدل های کلاس.
* **ساخت**
* ایجاد دانش نوشته شده به واحدهای قابل استقرار.
* در KIE این واحد های JAR فایل هستند.
* **آزمون**
* تست دانش KIE قبل از انتشار در برنامه کاربردی.
* **استقرار**
* فرستادن واحدهای قابل استقرار (JAR) به برنامه کاربردی که قرار است از آنها استفاده شود .
* در KIE برای اینکار از ابزار Maven استفاده می شود .
* **استفاده**
* فراخوانی فایل های Jar و ایجاد جلسه کاری [[68]](#footnote-68) با این منظور که برنامه کاربردی بتواند با آن تعامل داشته باشد .
* **اجرا**
* تعامل برنامه با session های ایجاد شده از طریق API.
* **مدیریت**
* مدیریت هر KieSession یا KieContainer.

### ساخت، استقرار، اجرا و استفاده

در نسخه 6 راه جدید برای ساخت پایگاه دانش ارائه شده است . یعنی بجای استفاده از روش های برنامه ای که در نسخه های قبلی از آن استفاده می شد . از مدل پیکربندی استفاده شده است . که البته ساختار قبلی جهت مجتمع سازی قابل استفاده است .

در این روش برای ساخت از Maven استفاده می شود .یک ماژول KIE یک ماژول جاوایی مبتنی بر maven است . که یک فایل متاداده با نام kmodule.xml دارد . این فایل اطلاعاتی مربوط به ساختار آن ماژول است .

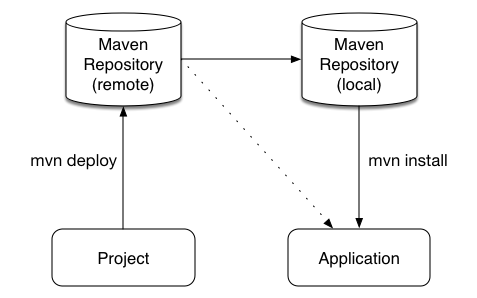
شکل زیر نمونه ای این ساختار پروژه و فایل را نشان میدهد .



شکل ‏3‑2 ) نمونه ای از ساختار پروژه و فایل POM

KIE از پیشفرض هایی برای حداقل کردن حجم تنظیمات استفاده می کند . بطوریکه یک فایل kmodule.xml خالی ساده ترین نوع پیکربندی است . ولی وجود آن الزامی است .

**Figure 4.3. Example project layout and Maven POM**



JARs can be deployed in one of two ways. Either added to the classpath, like any other JAR in a Maven dependency listing, or they can be dynamically loaded at runtime. KIE will scan the classpath to find all the JARs with a kmodule.xml in it. Each found JAR is represented by the KieModule interface. The terms classpath KieModule and dynamic KieModule are used to refer to the two loading approaches. While dynamic modules supports side by side versioning, classpath modules do not. Further once a module is on the classpath, no other version may be loaded dynamically.

Detailed references for the API are included in the next sections, the impatient can jump straight to the examples section, which is fairly self-explanatory on the different use cases.

### 4.2.2. Building

**Figure 4.4. org.kie.api.core.builder**



#### 4.2.2.1. Creating and building a Kie Project

A Kie Project has the structure of a normal Maven project with the only peculiarity of including a kmodule.xml file defining in a declaratively way the KieBases and KieSessions that can be created from it. This file has to be placed in the resources/META-INF folder of the Maven project while all the other Kie artifacts, such as DRL or a Excel files, must be stored in the resources folder or in any other subfolder under it.

Since meaningful defaults have been provided for all configuration aspects, the simplest kmodule.xml file can contain just an empty kmodule tag like the following:

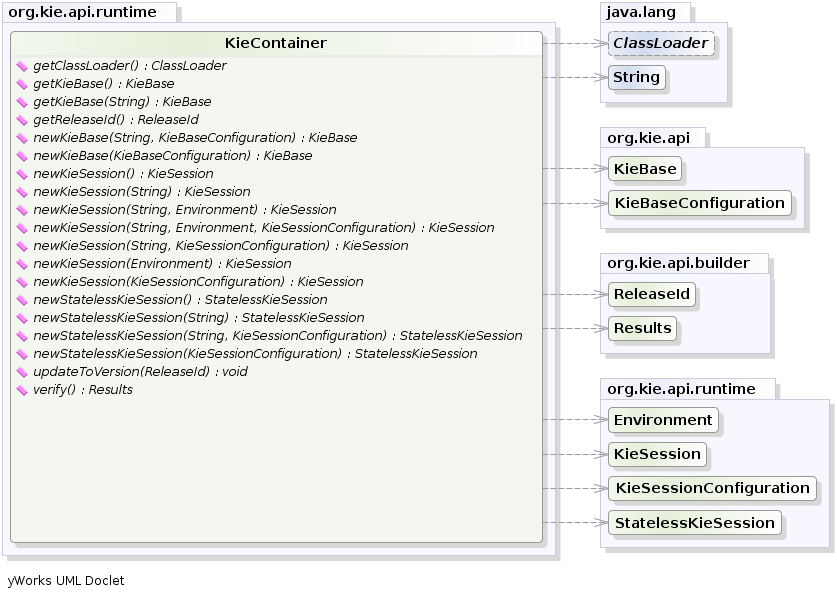
**Example 4.1. An empty kmodule.xml file**

**<?xml version="1.0" encoding="UTF-8"?>**

<kmodule xmlns="http://jboss.org/kie/6.0.0/kmodule"/>

In this way the kmodule will contain one single default KieBase. All Kie assets stored under the resources folder, or any of its subfolders, will be compiled and added to it. To trigger the building of these artifacts it is enough to create a KieContainer for them.

**Figure 4.5. KieContainer**



For this simple case it is enough to create a KieContainer that reads the files to be built from the classpath:

**Example 4.2. Creating a KieContainer from the classpath**

KieServices kieServices = KieServices.Factory.get();

KieContainer kContainer = kieServices.getKieClasspathContainer();

KieServices is the interface from where it possible to access all the Kie building and runtime facilities:

**Figure 4.6. KieServices**



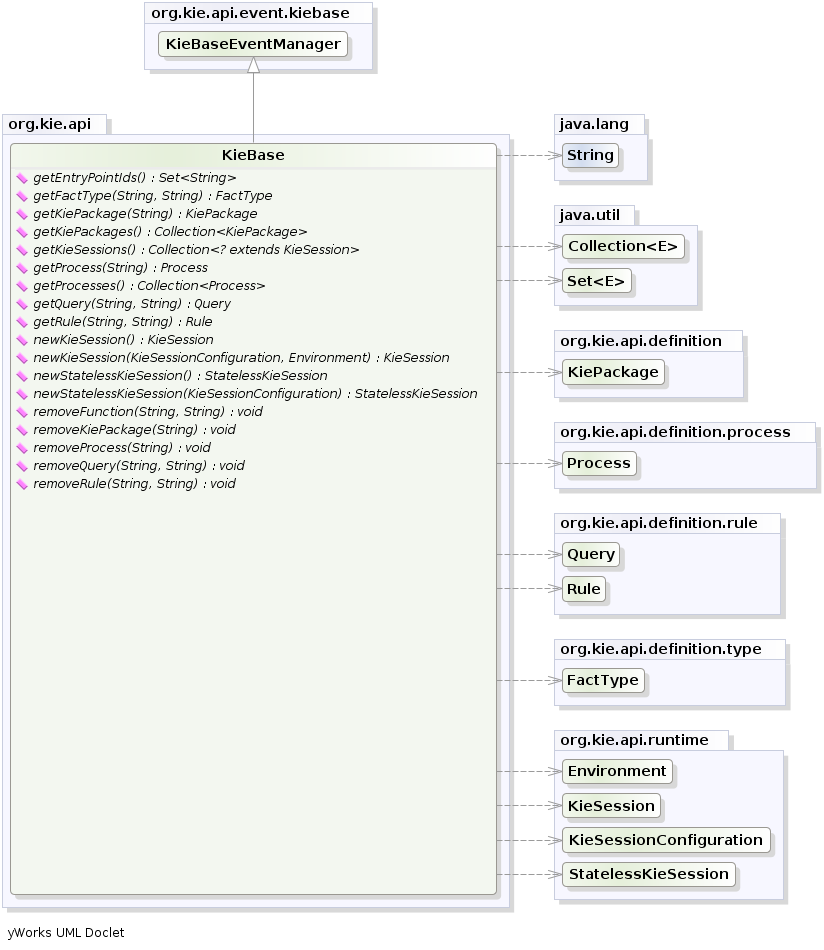
In this way all the Java sources and the Kie resources are compiled and deployed into the KieContainer which makes its contents available for use at runtime.

#### 4.2.2.2. The kmodule.xml file

As explained in the former section, the kmodule.xml file is the place where it is possible to declaratively configure the KieBase(s) and KieSession(s) that can be created from a KIE project.

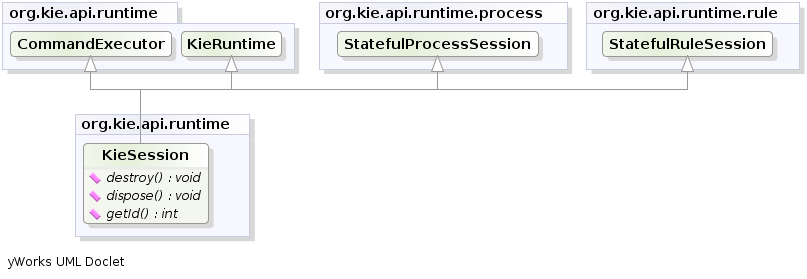
In particular a KieBase is a repository of all the application's knowledge definitions. It will contain rules, processes, functions, and type models. The KieBase itself does not contain data; instead, sessions are created from the KieBase into which data can be inserted and from which process instances may be started. Creating the KieBase can be heavy, whereas session creation is very light, so it is recommended that KieBase be cached where possible to allow for repeated session creation. However end-users usually shouldn't worry about it, because this caching mechanism is already automatically provided by the KieContainer.

**Figure 4.7. KieBase**



Conversely the KieSession stores and executes on the runtime data. It is created from the KieBase or more easily can be created directly from the KieContainer if it has been defined in the kmodule.xml file

**Figure 4.8. KieSession**



The kmodule.xml allows to define and configure one or more KieBases and for each KieBase all the differentKieSessions that can be created from it, as showed by the follwing example:

**Example 4.3. A sample kmodule.xml file**

<kmodule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://jboss.org/kie/6.0.0/kmodule">

<configuration>

<property key="drools.evaluator.supersetOf" value="org.mycompany.SupersetOfEvaluatorDefinition"/>

</configuration>

<kbase name="KBase1" default="true" eventProcessingMode="cloud" equalsBehavior="equality" declarativeAgenda="enabled" packages="org.domain.pkg1">

<ksession name="KSession2\_1" type="stateful" default="true"/>

<ksession name="KSession2\_2" type="stateless" default="false" beliefSystem="jtms"/>

</kbase>

<kbase name="KBase2" default="false" eventProcessingMode="stream" equalsBehavior="equality" declarativeAgenda="enabled" packages="org.domain.pkg2, org.domain.pkg3" includes="KBase1">

<ksession name="KSession3\_1" type="stateful" default="false" clockType="realtime">

<fileLogger file="drools.log" threaded="true" interval="10"/>

<workItemHandlers>

<workItemHandler name="name" type="org.domain.WorkItemHandler"/>

</workItemHandlers>

<listeners>

<ruleRuntimeEventListener type="org.domain.RuleRuntimeListener"/>

<agendaEventListener type="org.domain.FirstAgendaListener"/>

<agendaEventListener type="org.domain.SecondAgendaListener"/>

<processEventListener type="org.domain.ProcessListener"/>

</listeners>

</ksession>

</kbase>

</kmodule>

Here the <configuration> tag contains a list of key-value pairs that are the optional properties used to configure the KieBases building process. For instance this sample kmodule.xml file defines an additional custom operator named supersetOf and implemented by the org.mycompany.SupersetOfEvaluatorDefinitionclass.

After this 2 KieBases have been defined and it is possible to instance 2 different types of KieSessions from the first one, while only one from the second. A list of the attributes that can be defined on the kbase tag, together with their meaning and default values follows:

**Table 4.1. kbase Attributes**

| **Attribute name** | **Default value** | **Admitted values** | **Meaning** |
| --- | --- | --- | --- |
| name | none | any | The name with which retrieve this KieBase from the KieContainer. This is the only mandatory attribute. |
| includes | none | any comma separated list | A comma separated list of other KieBases contained in this kmodule. The artifacts of all these KieBases will be also included in this one. |
| packages | all | any comma separated list | By default all the Drools artifacts under the resources folder, at any level, are included into the KieBase. This attribute allows to limit the artifacts that will be compiled in this KieBase to only the ones belonging to the list of packages. |
| default | false | true, false | Defines if this KieBase is the default one for this module, so it can be created from the KieContainer without passing any name to it. There can be at most one default KieBase in each module. |
| equalsBehavior | identity | identity, equality | Defines the behavior of Drools when a new fact is inserted into the Working Memory. With identity it always create a new FactHandle unless the same object isn't already present in the Working Memory, while with equality only if the newly inserted object is not equal (according to its equal method) to an already existing fact. |
| eventProcessingMode | cloud | cloud, stream | When compiled in cloud mode the KieBase treats events as normal facts, while in stream mode allow temporal reasoning on them. |
| declarativeAgenda | disabled | disabled, enabled | Defines if the Declarative Agenda is enabled or not. |

Similarly all attributes of the ksession tag (except of course the name) have meaningful default. They are listed and described in the following table:

**Table 4.2. ksession Attributes**

| **Attribute name** | **Default value** | **Admitted values** | **Meaning** |
| --- | --- | --- | --- |
| name | none | any | Unique name of this KieSession. Used to fetch the KieSession from the KieContainer. This is the only mandatory attribute. |
| type | stateful | stateful, stateless | A stateful session allows to iteratively work with the Working Memory, while a stateless one is a one-off execution of a Working Memory with a provided data set. |
| default | false | true, false | Defines if this KieSession is the default one for this module, so it can be created from the KieContainer without passing any name to it. In each module there can be at most one default KieSession for each type. |
| clockType | realtime | realtime, pseudo | Defines if events timestamps are determined by the system clock or by a psuedo clock controlled by the application. This clock is specially useful for unit testing temporal rules. |
| beliefSystem | simple | simple, jtms, defeasible | Defines the type of belief system used by the KieSession. |

As outlined in the former kmodule.xml sample, it is also possible to declaratively create on each KieSession a file (or a console) logger, one or more WorkItemHandlers and some listeners that can be of 3 different types: ruleRuntimeEventListener, agendaEventListener and processEventListener

Having defined a kmodule.xml like the one in the former sample, it is now possible to simply retrieve the KieBases and KieSessions from the KieContainer using their names.

**Example 4.4. Retriving KieBases and KieSessions from the KieContainer**

KieServices kieServices = KieServices.Factory.get();

KieContainer kContainer = kieServices.getKieClasspathContainer();

KieBase kBase1 = kContainer.getKieBase("KBase1");

KieSession kieSession1 = kContainer.newKieSession("KSession2\_1");

StatelessKieSession kieSession2 = kContainer.newStatelessKieSession("KSession2\_2");

It has to be noted that since KSession2\_1 and KSession2\_2 are of 2 different types (the first is stateful, while the second is stateless) it is necessary to invoke 2 different methods on the KieContainer according to their declared type. If the type of the KieSession requested to the KieContainer doesn't correspond with the one declared in the kmodule.xml file the KieContainer will throw a RuntimeException. Also since a KieBase and aKieSession have been flagged as default is it possible to get them from the KieContainer without passing any name.

**Example 4.5. Retriving default KieBases and KieSessions from the KieContainer**

KieContainer kContainer = ...

KieBase kBase1 = kContainer.getKieBase(); *// returns KBase1*

KieSession kieSession1 = kContainer.newKieSession(); *// returns KSession2\_1*

Since a Kie project is also a Maven project the groupId, artifactId and version declared in the pom.xml file are used to generate a ReleaseId that uniquely identifies this project inside your application. This allows creation of a new KieContainer from the project by simply passing its ReleaseId to the KieServices.

**Example 4.6. Creating a KieContainer of an existing project by ReleaseId**

KieServices kieServices = KieServices.Factory.get();

ReleaseId releaseId = kieServices.newReleaseId( "org.acme", "myartifact", "1.0" );

KieContainer kieContainer = kieServices.newKieContainer( releaseId );

#### 4.2.2.3. Building with Maven

The KIE plugin for Maven ensures that artifact resources are validated and pre-compiled, it is recommended that this is used at all times. To use the plugin simply add it to the build section of the Maven pom.xml

**Example 4.7. Adding the KIE plugin to a Maven pom.xml**

<build>

<plugins>

<plugin>

<groupId>org.kie</groupId>

<artifactId>kie-maven-plugin</artifactId>

<version>${project.version}</version>

<extensions>true</extensions>

</plugin>

</plugins>

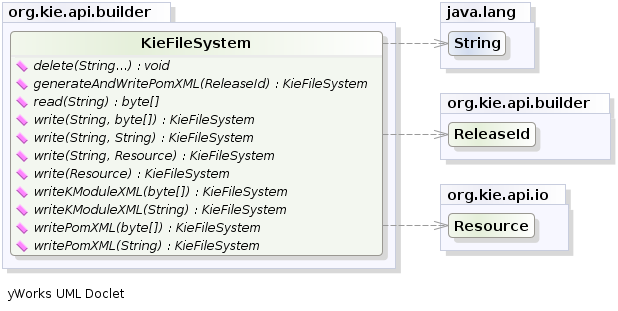
</build>

Building a KIE module without the Maven plugin will copy all the resources, as is, into the resulting JAR. When that JAR is loaded by the runtime, it will attempt to build all the resources then. If there are compilation issues it will return a null KieContainer. It also pushes the compilation overhead to the runtime. In general this is not recommended, and the Maven plugin should always be used.

#### 4.2.2.4. Defining a KieModule programmatically

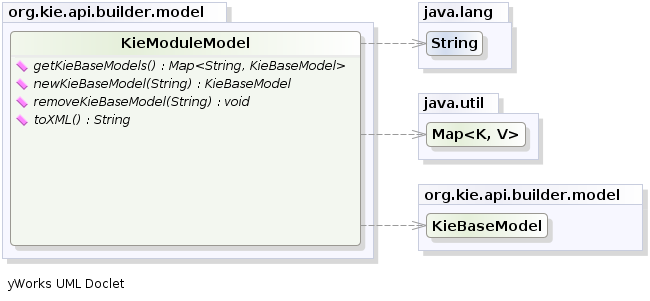
It is also possible to define the KieBases and KieSessions belonging to a KieModule programmatically instead of the declarative definition in the kmodule.xml file. The same programmatic API also allows in explicitly adding the file containing the Kie artifacts instead of automatically read them from the resources folder of your project. To do that it is necessary to create a KieFileSystem, a sort of virtual file system, and add all the resources contained in your project to it.

**Figure 4.9. KieFileSystem**



Like all other Kie core components you can obtain an instance of the KieFileSystem from the KieServices. The kmodule.xml configuration file must be added to the filesystem. This is a mandatory step. Kie also provides a convenient fluent API, implemented by the KieModuleModel, to programmatically create this file.

**Figure 4.10. KieModuleModel**



To do this in practice it is necessary to create a KieModuleModel from the KieServices, configure it with the desired KieBases and KieSessions, convert it in XML and add the XML to the KieFileSystem. This process is shown by the following example:

**Example 4.8. Creating a kmodule.xml programmatically and adding it to a KieFileSystem**

KieServices kieServices = KieServices.Factory.get();

KieModuleModel kieModuleModel = kieServices.newKieModuleModel();

KieBaseModel kieBaseModel1 = kieModuleModel.newKieBaseModel( "KBase1 ")

.setDefault( **true** )

.setEqualsBehavior( EqualityBehaviorOption.EQUALITY )

.setEventProcessingMode( EventProcessingOption.STREAM );

KieSessionModel ksessionModel1 = kieBaseModel1.newKieSessionModel( "KSession1" )

.setDefault( **true** )

.setType( KieSessionModel.KieSessionType.STATEFUL )

.setClockType( ClockTypeOption.get("realtime") );

KieFileSystem kfs = kieServices.newKieFileSystem();

kfs.writeKModuleXML(kieModuleModel.toXML());

At this point it is also necessary to add to the KieFileSystem, through its fluent API, all others Kie artifacts composing your project. These artifacts have to be added in the same position of a corresponding usual Maven project.

**Example 4.9. Adding Kie artifacts to a KieFileSystem**

KieFileSystem kfs = ...

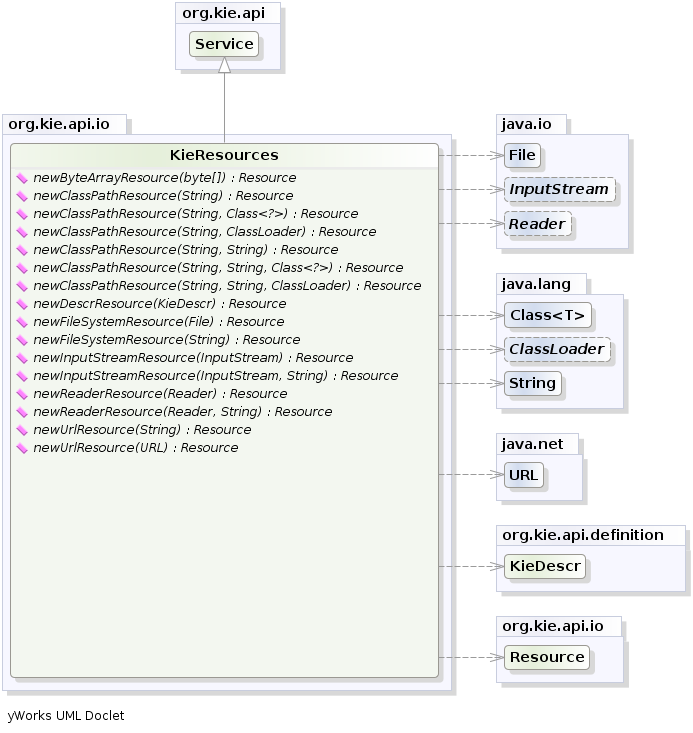
kfs.write( "src/main/resources/KBase1/ruleSet1.drl", stringContainingAValidDRL )

.write( "src/main/resources/dtable.xls",

kieServices.getResources().newInputStreamResource( dtableFileStream ) );

This example shows that it is possible to add the Kie artifacts both as plain Strings and as Resources. In the latter case the Resources can be created by the KieResources factory, also provided by the KieServices. TheKieResources provides many convenient factory methods to convert an InputStream, a URL, a File, or aString representing a path of your file system to a Resource that can be managed by the KieFileSystem.

**Figure 4.11. KieResources**



Normally the type of a Resource can be inferred from the extension of the name used to add it to theKieFileSystem. However it also possible to not follow the Kie conventions about file extensions and explicitly assign a specific ResourceType to a Resource as shown below:

**Example 4.10. Creating and adding a Resource with an explicit type**

KieFileSystem kfs = ...

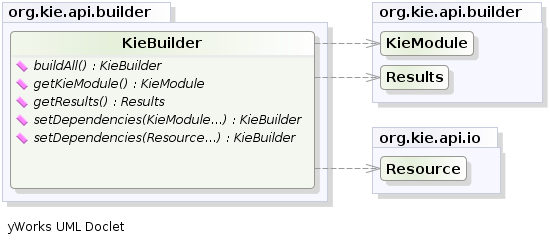
kfs.write( "src/main/resources/myDrl.txt",

kieServices.getResources().newInputStreamResource( drlStream )

.setResourceType(ResourceType.DRL) );

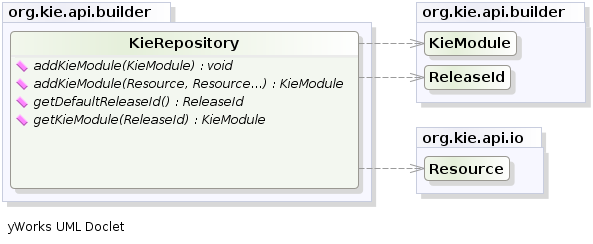
Add all the resources to the KieFileSystem and build it by passing the KieFileSystem to a KieBuilder

**Figure 4.12. KieBuilder**



When the contents of a KieFileSystem are successfully built, the resulting KieModule is automatically added to the KieRepository. The KieRepository is a singleton acting as a repository for all the available KieModules.

**Figure 4.13. KieRepository**



After this it is possible to create through the KieServices a new KieContainer for that KieModule using itsReleaseId. However, since in this case the KieFileSystem doesn't contain any pom.xml file (it is possible to add one using the KieFileSystem.writePomXML method), Kie cannot determine the ReleaseId of the KieModuleand assign to it a default one. This default ReleaseId can be obtained from the KieRepository and used to identify the KieModule inside the KieRepository itself. The following example shows this whole process.

**Example 4.11. Building the contents of a KieFileSystem and creating a KieContainer**

KieServices kieServices = KieServices.Factory.get();

KieFileSystem kfs = ...

kieServices.newKieBuilder( kfs ).buildAll();

KieContainer kieContainer = kieServices.newKieContainer(kieServices.getRepository().getDefaultReleaseId());

At this point it is possible to get KieBases and create new KieSessions from this KieContainer exactly in the same way as in the case of a KieContainer created directly from the classpath.

It is a best practice to check the compilation results. The KieBuilder reports compilation results of 3 different severities: ERROR, WARNING and INFO. An ERROR indicates that the compilation of the project failed and in the case no KieModule is produced and nothing is added to the KieRepository. WARNING and INFO results can be ignored, but are available for inspection.

**Example 4.12. Checking that a compilation didn't produce any error**

KieBuilder kieBuilder = kieServices.newKieBuilder( kfs ).buildAll();

assertEquals( 0, kieBuilder.getResults().getMessages( Message.Level.ERROR ).size() );

#### 4.2.2.5. Changing the Default Build Result Severity

In some cases, it is possible to change the default severity of a type of build result. For instance, when a new rule with the same name of an existing rule is added to a package, the default behavior is to replace the old rule by the new rule and report it as an INFO. This is probably ideal for most use cases, but in some deployments the user might want to prevent the rule update and report it as an error.

Changing the default severity for a result type, configured like any other option in Drools, can be done by API calls, system properties or configuration files. As of this version, Drools supports configurable result severity for rule updates and function updates. To configure it using system properties or configuration files, the user has to use the following properties:

**Example 4.13. Setting the severity using properties**

// sets the severity of rule updates

drools.kbuilder.severity.duplicateRule = <INFO|WARNING|ERROR>

// sets the severity of function updates

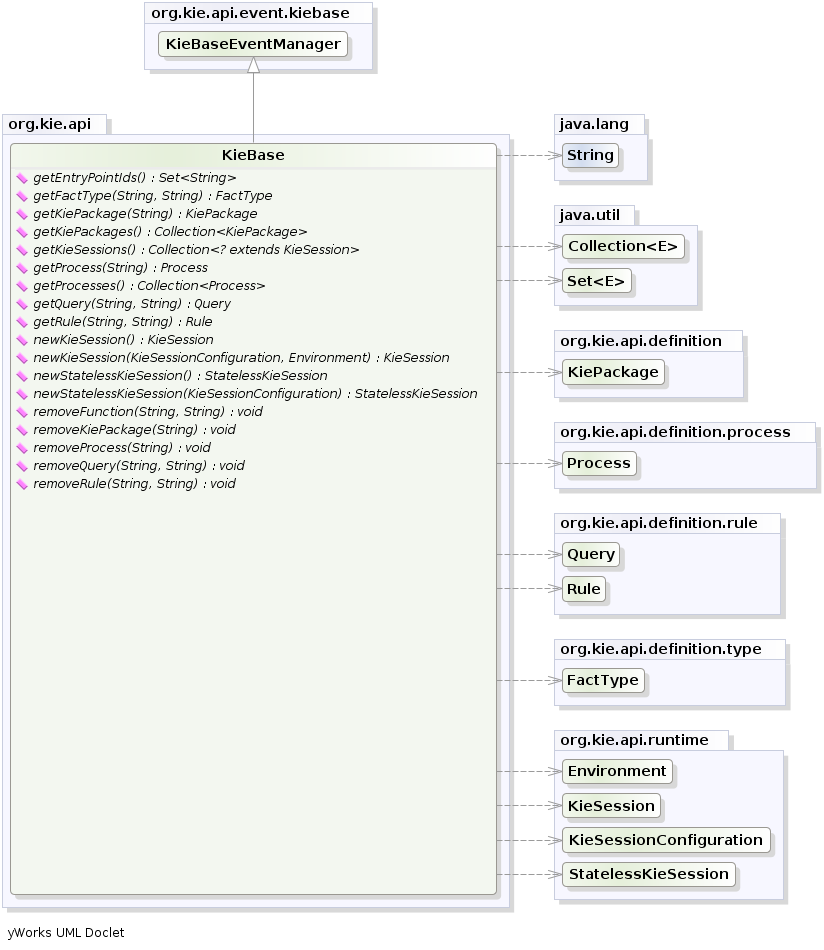
drools.kbuilder.severity.duplicateFunction = <INFO|WARNING|ERROR>

### 4.2.3. Deploying

#### 4.2.3.1. KieBase

The KieBase is a repository of all the application's knowledge definitions. It will contain rules, processes, functions, and type models. The KieBase itself does not contain data; instead, sessions are created from theKieBase into which data can be inserted and from which process instances may be started. The KieBase can be obtained from the KieContainer containing the KieModule where the KieBase has been defined.

**Figure 4.14. KieBase**



Sometimes, for instance in a OSGi environment, the KieBase needs to resolve types that are not in the default class loader. In this case it will be necessary to create a KieBaseConfiguration with an additional class loader and pass it to KieContainer when creating a new KieBase from it.

**Example 4.14. Creating a new KieBase with a custom ClassLoader**

KieServices kieServices = KieServices.Factory.get();

KieBaseConfiguration kbaseConf = kieServices.newKieBaseConfiguration( **null**, MyType.class.getClassLoader() );

KieBase kbase = kieContainer.newKieBase( kbaseConf );

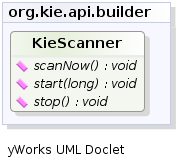
#### 4.2.3.2. KieSessions and KieBase Modifications

KieSessions will be discussed in more detail in section "Running". The KieBase creates and returns KieSessionobjects, and it may optionally keep references to those. When KieBase modifications occur those modifications are applied against the data in the sessions. This reference is a weak reference and it is also optional, which is controlled by a boolean flag.

#### 4.2.3.3. KieScanner

The KieScanner allows continuous monitoring of your Maven repository to check whether a new release of a Kie project has been installed. A new release is deployed in the KieContainer wrapping that project. The use of the KieScanner requires kie-ci.jar to be on the classpath.

**Figure 4.15. KieScanner**



A KieScanner can be registered on a KieContainer as in the following example.

**Example 4.15. Registering and starting a KieScanner on a KieContainer**

KieServices kieServices = KieServices.Factory.get();

ReleaseId releaseId = kieServices.newReleaseId( "org.acme", "myartifact", "1.0-SNAPSHOT" );

KieContainer kContainer = kieServices.newKieContainer( releaseId );

KieScanner kScanner = kieServices.newKieScanner( kContainer );

*// Start the KieScanner polling the Maven repository every 10 seconds*

kScanner.start( 10000L );

In this example the KieScanner is configured to run with a fixed time interval, but it is also possible to run it on demand by invoking the scanNow() method on it. If the KieScanner finds in the Maven repository an updated version of the Kie project used by that KieContainer it automatically downloads the new version and triggers an incremental build of the new project. From this moment all the new KieBases and KieSessions created from that KieContainer will use the new project version.

The KieScanner will only pickup changes to deployed jars if it is using a SNAPSHOT, version range, the LATEST, or the RELEASE setting. Fixed versions will not automatically update at runtime.

#### 4.2.3.4. Maven Versions and Dependencies

Maven supports a number of mechanisms to manage versioning and dependencies within applications. Modules can be published with specific version numbers, or they can use the SNAPSHOT suffix. Dependencies can specify version ranges to consume, or take avantage of SNAPSHOT mechanism.

StackOverflow provides a very good description for this, which is reproduced below.

<http://stackoverflow.com/questions/30571/how-do-i-tell-maven-to-use-the-latest-version-of-a-dependency>

If you always want to use the newest version, Maven has two keywords you can use as an alternative to version ranges. You should use these options with care as you are no longer in control of the plugins/dependencies you are using.

When you depend on a plugin or a dependency, you can use the a version value of LATEST or RELEASE. LATEST refers to the latest released or snapshot version of a particular artifact, the most recently deployed artifact in a particular repository. RELEASE refers to the last non-snapshot release in the repository. In general, it is not a best practice to design software which depends on a non-specific version of an artifact. If you are developing software, you might want to use RELEASE or LATEST as a convenience so that you don't have to update version numbers when a new release of a third-party library is released. When you release software, you should always make sure that your project depends on specific versions to reduce the chances of your build or your project being affected by a software release not under your control. Use LATEST and RELEASE with caution, if at all.

See the POM Syntax section of the Maven book for more details.

<http://books.sonatype.com/mvnref-book/reference/pom-relationships-sect-pom-syntax.html>

<http://books.sonatype.com/mvnref-book/reference/pom-relationships-sect-project-dependencies.html>

Here's an example illustrating the various options. In the Maven repository, com.foo:my-foo has the following metadata:

<metadata>

<groupId>com.foo</groupId>

<artifactId>my-foo</artifactId>

<version>2.0.0</version>

<versioning>

<release>1.1.1</release>

<versions>

<version>1.0</version>

<version>1.0.1</version>

<version>1.1</version>

<version>1.1.1</version>

<version>2.0.0</version>

</versions>

<lastUpdated>20090722140000</lastUpdated>

</versioning>

</metadata>

If a dependency on that artifact is required, you have the following options (other version ranges can be specified of course, just showing the relevant ones here): Declare an exact version (will always resolve to 1.0.1):

<version>[1.0.1]</version>

Declare an explicit version (will always resolve to 1.0.1 unless a collision occurs, when Maven will select a matching version):

<version>1.0.1</version>

Declare a version range for all 1.x (will currently resolve to 1.1.1):

<version>[1.0.0,2.0.0)</version>

Declare an open-ended version range (will resolve to 2.0.0):

<version>[1.0.0,)</version>

Declare the version as LATEST (will resolve to 2.0.0):

<version>LATEST</version>

Declare the version as RELEASE (will resolve to 1.1.1):

<version>RELEASE</version>

Note that by default your own deployments will update the "latest" entry in the Maven metadata, but to update the "release" entry, you need to activate the "release-profile" from the Maven super POM. You can do this with either "-Prelease-profile" or "-DperformRelease=true"

#### 4.2.3.5. Settings.xml and Remote Repository Setup

The maven settings.xml is used to configure Maven execution. Detailed instructions can be found at the Maven website:

<http://maven.apache.org/settings.html>

The settings.xml file can be located in 3 locations, the actual settings used is a merge of those 3 locations.

* The Maven install: $M2\_HOME/conf/settings.xml
* A user's install: ${user.home}/.m2/settings.xml
* Folder location specified by the system property kie.maven.settings.custom

The settings.xml is used to specify the location of remote repositories. It is important that you activate the profile that specifies the remote repository, typically this can be done using "activeByDefault":

<profiles>

<profile>

<id>profile-1</id>

<activation>

<activeByDefault>true</activeByDefault>

</activation>

...

</profile>

</profiles>

Maven provides detailed documentation on using multiple remote repositories:

<http://maven.apache.org/guides/mini/guide-multiple-repositories.html>

### 4.2.4. Running

#### 4.2.4.1. KieBase

The KieBase is a repository of all the application's knowledge definitions. It will contain rules, processes, functions, and type models. The KieBase itself does not contain data; instead, sessions are created from theKieBase into which data can be inserted and from which process instances may be started. The KieBase can be obtained from the KieContainer containing the KieModule where the KieBase has been defined.

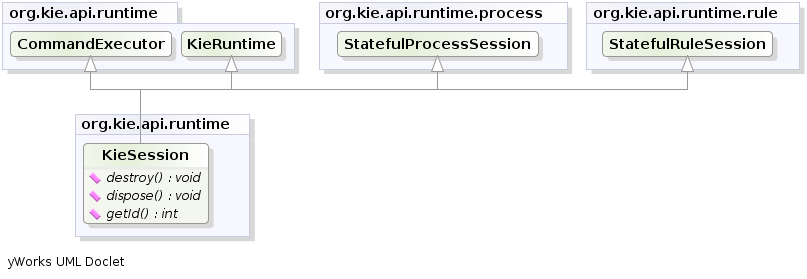
**Example 4.16. Getting a KieBase from a KieContainer**

KieBase kBase = kContainer.getKieBase();

#### 4.2.4.2. KieSession

The KieSession stores and executes on the runtime data. It is created from the KieBase.

**Figure 4.16. KieSession**



**Example 4.17. Create a KieSession from a KieBase**

KieSession ksession = kbase.newKieSession();

#### 4.2.4.3. KieRuntime

##### 4.2.4.3.1. KieRuntime

The KieRuntime provides methods that are applicable to both rules and processes, such as setting globals and registering channels. ("Exit point" is an obsolete synonym for "channel".)

**Figure 4.17. KieRuntime**



###### 4.2.4.3.1.1. Globals

Globals are named objects that are made visible to the rule engine, but in a way that is fundamentally different from the one for facts: changes in the object backing a global do not trigger reevaluation of rules. Still, globals are useful for providing static information, as an object offering services that are used in the RHS of a rule, or as a means to return objects from the rule engine. When you use a global on the LHS of a rule, make sure it is immutable, or, at least, don't expect changes to have any effect on the behavior of your rules.

A global must be declared in a rules file, and then it needs to be backed up with a Java object.

global java.util.List list

With the Knowledge Base now aware of the global identifier and its type, it is now possible to callksession.setGlobal() with the global's name and an object, for any session, to associate the object with the global. Failure to declare the global type and identifier in DRL code will result in an exception being thrown from this call.

List list = **new** ArrayList();

ksession.setGlobal("list", list);

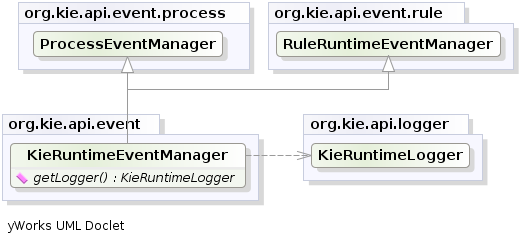
Make sure to set any global before it is used in the evaluation of a rule. Failure to do so results in aNullPointerException.

#### 4.2.4.4. Event Model

The event package provides means to be notified of rule engine events, including rules firing, objects being asserted, etc. This allows separation of logging and auditing activities from the main part of your application (and the rules).

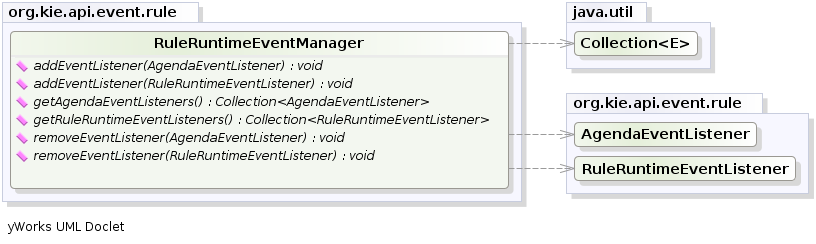
The KieRuntimeEventManager interface is implemented by the KieRuntime which provides two interfaces,RuleRuntimeEventManager and ProcessEventManager. We will only cover the RuleRuntimeEventManager here.

**Figure 4.18. KieRuntimeEventManager**



The RuleRuntimeEventManager allows for listeners to be added and removed, so that events for the working memory and the agenda can be listened to.

**Figure 4.19. RuleRuntimeEventManager**



The following code snippet shows how a simple agenda listener is declared and attached to a session. It will print matches after they have fired.

**Example 4.18. Adding an AgendaEventListener**

ksession.addEventListener( **new** DefaultAgendaEventListener() {

**public** **void** **afterMatchFired**(AfterMatchFiredEvent event) {

**super**.afterMatchFired( event );

System.out.println( event );

}

});

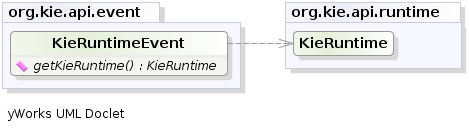
Drools also provides DebugRuleRuntimeEventListener and DebugAgendaEventListener which implement each method with a debug print statement. To print all Working Memory events, you add a listener like this:

**Example 4.19. Adding a DebugRuleRuntimeEventListener**

ksession.addEventListener( **new** DebugRuleRuntimeEventListener() );

All emitted events implement the KieRuntimeEvent interface which can be used to retrieve the actualKnowlegeRuntime the event originated from.

**Figure 4.20. KieRuntimeEvent**



The events currently supported are:

* MatchCreatedEvent
* MatchCancelledEvent
* BeforeMatchFiredEvent
* AfterMatchFiredEvent
* AgendaGroupPushedEvent
* AgendaGroupPoppedEvent
* ObjectInsertEvent
* ObjectDeletedEvent
* ObjectUpdatedEvent
* ProcessCompletedEvent
* ProcessNodeLeftEvent
* ProcessNodeTriggeredEvent
* ProcessStartEvent

#### 4.2.4.5. KieRuntimeLogger

The KieRuntimeLogger uses the comprehensive event system in Drools to create an audit log that can be used to log the execution of an application for later inspection, using tools such as the Eclipse audit viewer.

**Figure 4.21. KieLoggers**



**Example 4.20. FileLogger**

KieRuntimeLogger logger =

KieServices.Factory.get().getLoggers().newFileLogger(ksession, "logdir/mylogfile");

...

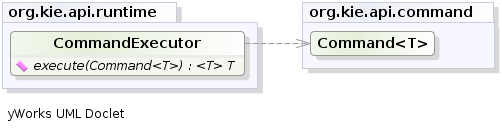
logger.close();

#### 4.2.4.6. Commands and the CommandExecutor

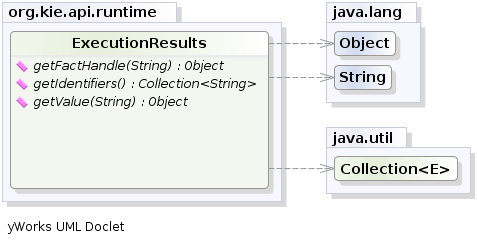
KIE has the concept of stateful or stateless sessions. Stateful sessions have already been covered, which use the standard KieRuntime, and can be worked with iteratively over time. Stateless is a one-off execution of a KieRuntime with a provided data set. It may return some results, with the session being disposed at the end, prohibiting further iterative interactions. You can think of stateless as treating an engine like a function call with optional return results.

The foundation for this is the CommandExecutor interface, which both the stateful and stateless interfaces extend. This returns an ExecutionResults:

**Figure 4.22. CommandExecutor**



**Figure 4.23. ExecutionResults**



The CommandExecutor allows for commands to be executed on those sessions, the only difference being that the StatelessKieSession executes fireAllRules() at the end before disposing the session. The commands can be created using the CommandExecutor .The Javadocs provide the full list of the allowed comands using theCommandExecutor.

setGlobal and getGlobal are two commands relevant to both Drools and jBPM.

Set Global calls setGlobal underneath. The optional boolean indicates whether the command should return the global's value as part of the ExecutionResults. If true it uses the same name as the global name. A String can be used instead of the boolean, if an alternative name is desired.

**Example 4.21. Set Global Command**

StatelessKieSession ksession = kbase.newStatelessKieSession();

ExecutionResults bresults =

ksession.execute( CommandFactory.newSetGlobal( "stilton", **new** Cheese( "stilton" ), **true**);

Cheese stilton = bresults.getValue( "stilton" );

Allows an existing global to be returned. The second optional String argument allows for an alternative return name.

**Example 4.22. Get Global Command**

StatelessKieSession ksession = kbase.newStatelessKieSession();

ExecutionResults bresults =

ksession.execute( CommandFactory.getGlobal( "stilton" );

Cheese stilton = bresults.getValue( "stilton" );

All the above examples execute single commands. The BatchExecution represents a composite command, created from a list of commands. It will iterate over the list and execute each command in turn. This means you can insert some objects, start a process, call fireAllRules and execute a query, all in a single execute(...)call, which is quite powerful.

The StatelessKieSession will execute fireAllRules() automatically at the end. However the keen-eyed reader probably has already noticed the FireAllRules command and wondered how that works with a StatelessKieSession. The FireAllRules command is allowed, and using it will disable the automatic execution at the end; think of using it as a sort of manual override function.

Any command, in the batch, that has an out identifier set will add its results to the returnedExecutionResults instance. Let's look at a simple example to see how this works. The example presented includes command from the Drools and jBPM, for the sake of illustration. They are covered in more detail in the Drool and jBPM specific sections.

**Example 4.23. BatchExecution Command**

StatelessKieSession ksession = kbase.newStatelessKieSession();

List cmds = **new** ArrayList();

cmds.add( CommandFactory.newInsertObject( **new** Cheese( "stilton", 1), "stilton") );

cmds.add( CommandFactory.newStartProcess( "process cheeses" ) );

cmds.add( CommandFactory.newQuery( "cheeses" ) );

ExecutionResults bresults = ksession.execute( CommandFactory.newBatchExecution( cmds ) );

Cheese stilton = ( Cheese ) bresults.getValue( "stilton" );

QueryResults qresults = ( QueryResults ) bresults.getValue( "cheeses" );

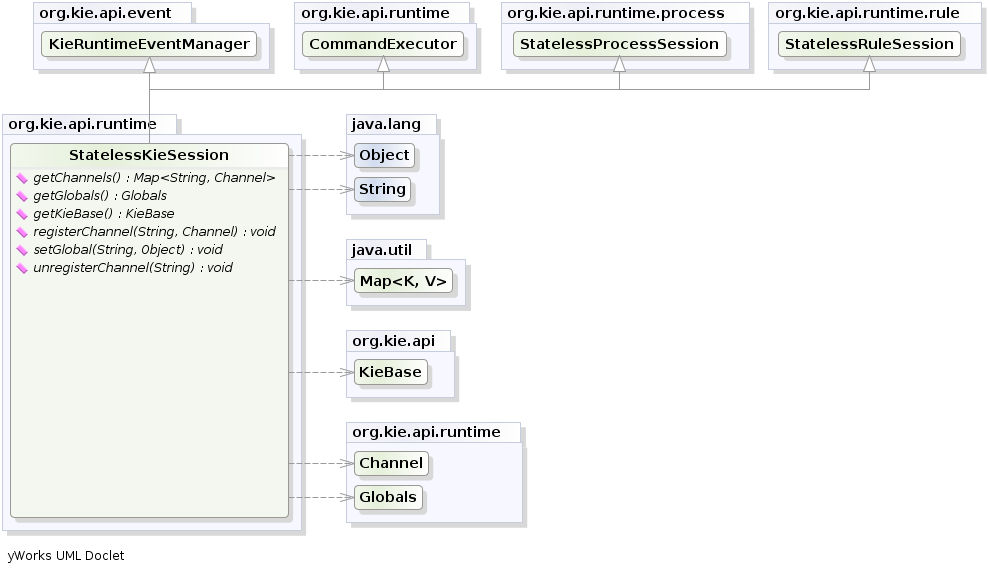
In the above example multiple commands are executed, two of which populate the ExecutionResults. The query command defaults to use the same identifier as the query name, but it can also be mapped to a different identifier.

All commands support XML and jSON marshalling using XStream, as well as JAXB marshalling. This is covered in section [Commands API](https://docs.jboss.org/drools/release/latest/drools-docs/html/ch11.html#CommandsAPISection).

#### 4.2.4.7. StatelessKieSession

The StatelessKieSession wraps the KieSession, instead of extending it. Its main focus is on the decision service type scenarios. It avoids the need to call dispose(). Stateless sessions do not support iterative insertions and the method call fireAllRules() from Java code; the act of calling execute() is a single-shot method that will internally instantiate a KieSession, add all the user data and execute user commands, callfireAllRules(), and then call dispose(). While the main way to work with this class is via the BatchExecution(a subinterface of Command) as supported by the CommandExecutor interface, two convenience methods are provided for when simple object insertion is all that's required. The CommandExecutor and BatchExecution are talked about in detail in their own section.

**Figure 4.24. StatelessKieSession**



Our simple example shows a stateless session executing a given collection of Java objects using the convenience API. It will iterate the collection, inserting each element in turn.

**Example 4.24. Simple StatelessKieSession execution with a Collection**

StatelessKieSession ksession = kbase.newStatelessKieSession();

ksession.execute( collection );

If this was done as a single Command it would be as follows:

**Example 4.25. Simple StatelessKieSession execution with InsertElements Command**

ksession.execute( CommandFactory.newInsertElements( collection ) );

If you wanted to insert the collection itself, and the collection's individual elements, thenCommandFactory.newInsert(collection) would do the job.

Methods of the CommandFactory create the supported commands, all of which can be marshalled using XStream and the BatchExecutionHelper. BatchExecutionHelper provides details on the XML format as well as how to use Drools Pipeline to automate the marshalling of BatchExecution and ExecutionResults.

StatelessKieSession supports globals, scoped in a number of ways. We cover the non-command way first, as commands are scoped to a specific execution call. Globals can be resolved in three ways.

* The StatelessKieSession method getGlobals() returns a Globals instance which provides access to the session's globals. These are shared for all execution calls. Exercise caution regarding mutable globals because execution calls can be executing simultaneously in different threads.

**Example 4.26. Session scoped global**

StatelessKieSession ksession = kbase.newStatelessKieSession();

*// Set a global hbnSession, that can be used for DB interactions in the rules.*

ksession.setGlobal( "hbnSession", hibernateSession );

*// Execute while being able to resolve the "hbnSession" identifier.*

ksession.execute( collection );

* Using a delegate is another way of global resolution. Assigning a value to a global (withsetGlobal(String, Object)) results in the value being stored in an internal collection mapping identifiers to values. Identifiers in this internal collection will have priority over any supplied delegate. Only if an identifier cannot be found in this internal collection, the delegate global (if any) will be used.
* The third way of resolving globals is to have execution scoped globals. Here, a Command to set a global is passed to the CommandExecutor.

The CommandExecutor interface also offers the ability to export data via "out" parameters. Inserted facts, globals and query results can all be returned.

**Example 4.27. Out identifiers**

*// Set up a list of commands*

List cmds = **new** ArrayList();

cmds.add( CommandFactory.newSetGlobal( "list1", **new** ArrayList(), **true** ) );

cmds.add( CommandFactory.newInsert( **new** Person( "jon", 102 ), "person" ) );

cmds.add( CommandFactory.newQuery( "Get People" "getPeople" );

*// Execute the list*

ExecutionResults results =

ksession.execute( CommandFactory.newBatchExecution( cmds ) );

*// Retrieve the ArrayList*

results.getValue( "list1" );

*// Retrieve the inserted Person fact*

results.getValue( "person" );

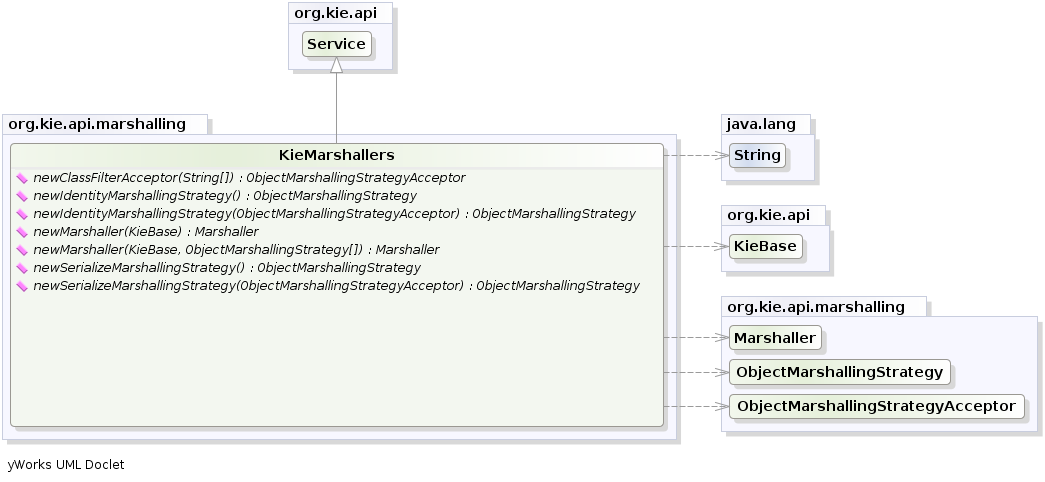
*// Retrieve the query as a QueryResults instance.*

results.getValue( "Get People" );

#### 4.2.4.8. Marshalling

The KieMarshallers are used to marshal and unmarshal KieSessions.

**Figure 4.25. KieMarshallers**



An instance of the KieMarshallers can be retrieved from the KieServices. A simple example is shown below:

**Example 4.28. Simple Marshaller Example**

*// ksession is the KieSession*

*// kbase is the KieBase*

ByteArrayOutputStream baos = **new** ByteArrayOutputStream();

Marshaller marshaller = KieServices.Factory.get().getMarshallers().newMarshaller( kbase );

marshaller.marshall( baos, ksession );

baos.close();

However, with marshalling, you will need more flexibility when dealing with referenced user data. To achieve this use the ObjectMarshallingStrategy interface. Two implementations are provided, but users can implement their own. The two supplied strategies are IdentityMarshallingStrategy andSerializeMarshallingStrategy. SerializeMarshallingStrategy is the default, as shown in the example above, and it just calls the Serializable or Externalizable methods on a user instance.IdentityMarshallingStrategy creates an integer id for each user object and stores them in a Map, while the id is written to the stream. When unmarshalling it accesses the IdentityMarshallingStrategy map to retrieve the instance. This means that if you use the IdentityMarshallingStrategy, it is stateful for the life of the Marshaller instance and will create ids and keep references to all objects that it attempts to marshal. Below is the code to use an Identity Marshalling Strategy.

**Example 4.29. IdentityMarshallingStrategy**

ByteArrayOutputStream baos = **new** ByteArrayOutputStream();

KieMarshallers kMarshallers = KieServices.Factory.get().getMarshallers()

ObjectMarshallingStrategy oms = kMarshallers.newIdentityMarshallingStrategy()

Marshaller marshaller =

kMarshallers.newMarshaller( kbase, **new** ObjectMarshallingStrategy[]{ oms } );

marshaller.marshall( baos, ksession );

baos.close();

Im most cases, a single strategy is insufficient. For added flexibility, the ObjectMarshallingStrategyAcceptorinterface can be used. This Marshaller has a chain of strategies, and while reading or writing a user object it iterates the strategies asking if they accept responsibility for marshalling the user object. One of the provided implementations is ClassFilterAcceptor. This allows strings and wild cards to be used to match class names. The default is "\*.\*", so in the above example the Identity Marshalling Strategy is used which has a default "\*.\*" acceptor.

Assuming that we want to serialize all classes except for one given package, where we will use identity lookup, we could do the following:

**Example 4.30. IdentityMarshallingStrategy with Acceptor**

ByteArrayOutputStream baos = **new** ByteArrayOutputStream();

KieMarshallers kMarshallers = KieServices.Factory.get().getMarshallers()

ObjectMarshallingStrategyAcceptor identityAcceptor =

kMarshallers.newClassFilterAcceptor( **new** String[] { "org.domain.pkg1.\*" } );

ObjectMarshallingStrategy identityStrategy =

kMarshallers.newIdentityMarshallingStrategy( identityAcceptor );

ObjectMarshallingStrategy sms = kMarshallers.newSerializeMarshallingStrategy();

Marshaller marshaller =

kMarshallers.newMarshaller( kbase,

**new** ObjectMarshallingStrategy[]{ identityStrategy, sms } );

marshaller.marshall( baos, ksession );

baos.close();

Note that the acceptance checking order is in the natural order of the supplied elements.

Also note that if you are using scheduled matches (i.e. some of your rules use timers or calendars) they are marshallable only if, before you use it, you configure your KieSession to use a trackable timer job factory manager as follows:

**Example 4.31. Configuring a trackable timer job factory manager**

KieSessionConfiguration ksconf = KieServices.Factory.get().newKieSessionConfiguration();

ksconf.setOption(TimerJobFactoryOption.get("trackable"));

KSession ksession = kbase.newKieSession(ksconf, **null**);

#### 4.2.4.9. Persistence and Transactions

Longterm out of the box persistence with Java Persistence API (JPA) is possible with Drools. It is necessary to have some implementation of the Java Transaction API (JTA) installed. For development purposes the Bitronix Transaction Manager is suggested, as it's simple to set up and works embedded, but for production use JBoss Transactions is recommended.

**Example 4.32. Simple example using transactions**

KieServices kieServices = KieServices.Factory.get();

Environment env = kieServices.newEnvironment();

env.set( EnvironmentName.ENTITY\_MANAGER\_FACTORY,

Persistence.createEntityManagerFactory( "emf-name" ) );

env.set( EnvironmentName.TRANSACTION\_MANAGER,

TransactionManagerServices.getTransactionManager() );

*// KieSessionConfiguration may be null, and a default will be used*

KieSession ksession =

kieServices.getStoreServices().newKieSession( kbase, **null**, env );

**int** sessionId = ksession.getId();

UserTransaction ut =

(UserTransaction) **new** InitialContext().lookup( "java:comp/UserTransaction" );

ut.begin();

ksession.insert( data1 );

ksession.insert( data2 );

ksession.startProcess( "process1" );

ut.commit();

To use a JPA, the Environment must be set with both the EntityManagerFactory and the TransactionManager. If rollback occurs the ksession state is also rolled back, hence it is possible to continue to use it after a rollback. To load a previously persisted KieSession you'll need the id, as shown below:

**Example 4.33. Loading a KieSession**

KieSession ksession =

kieServices.getStoreServices().loadKieSession( sessionId, kbase, **null**, env );

To enable persistence several classes must be added to your persistence.xml, as in the example below:

**Example 4.34. Configuring JPA**

<persistence-unit name="org.drools.persistence.jpa" transaction-type="JTA">

<provider>org.hibernate.ejb.HibernatePersistence</provider>

<jta-data-source>jdbc/BitronixJTADataSource</jta-data-source>

<class>org.drools.persistence.info.SessionInfo</class>

<class>org.drools.persistence.info.WorkItemInfo</class>

<properties>

<property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>

<property name="hibernate.max\_fetch\_depth" value="3"/>

<property name="hibernate.hbm2ddl.auto" value="update" />

<property name="hibernate.show\_sql" value="true" />

<property name="hibernate.transaction.manager\_lookup\_class"

value="org.hibernate.transaction.BTMTransactionManagerLookup" />

</properties>

</persistence-unit>

The jdbc JTA data source would have to be configured first. Bitronix provides a number of ways of doing this, and its documentation should be consulted for details. For a quick start, here is the programmatic approach:

**Example 4.35. Configuring JTA DataSource**

PoolingDataSource ds = **new** PoolingDataSource();

ds.setUniqueName( "jdbc/BitronixJTADataSource" );

ds.setClassName( "org.h2.jdbcx.JdbcDataSource" );

ds.setMaxPoolSize( 3 );

ds.setAllowLocalTransactions( **true** );

ds.getDriverProperties().put( "user", "sa" );

ds.getDriverProperties().put( "password", "sasa" );

ds.getDriverProperties().put( "URL", "jdbc:h2:mem:mydb" );

ds.init();

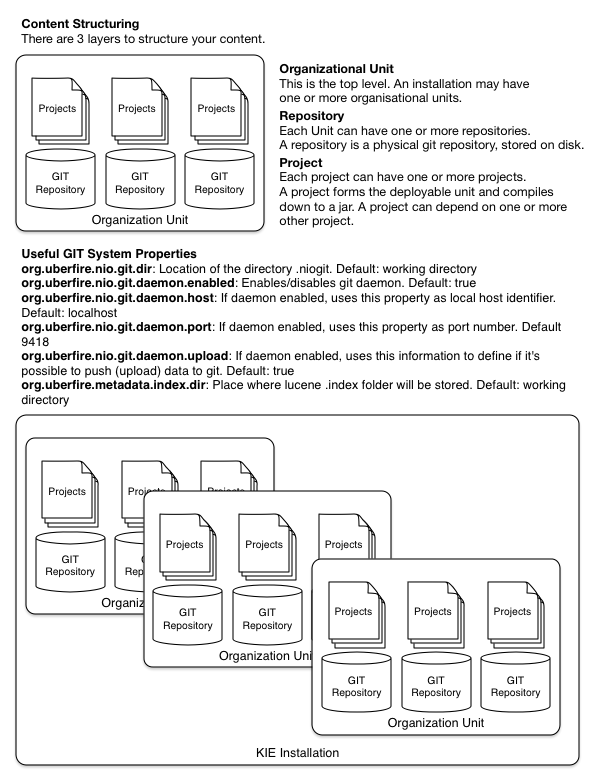
Bitronix also provides a simple embedded JNDI service, ideal for testing. To use it, add a jndi.properties file to your META-INF folder and add the following line to it:

**Example 4.36. JNDI properties**

java.naming.factory.initial=bitronix.tm.jndi.BitronixInitialContextFactory

### 4.2.5. Installation and Deployment Cheat Sheets

**Figure 4.26. Installation Overview**



**Figure 4.27. Deployment Overview**

### 4.2.6. Build, Deploy and Utilize Examples

The best way to learn the new build system is by example. The source project "drools-examples-api" contains a number of examples, and can be found at GitHub:

<https://github.com/droolsjbpm/drools/tree/6.0.x/drools-examples-api>

Each example is described below, the order starts with the simplest (most of the options are defaulted) and working its way up to more complex use cases.

The Deploy use cases shown below all involve mvn install. Remote deployment of JARs in Maven is well covered in Maven literature. Utilize refers to the initial act of loading the resources and providing access to the KIE runtimes. Where as Run refers to the act of interacting with those runtimes.

#### 4.2.6.1. Default KieSession

* Project: default-kesession.
* Summary: Empty kmodule.xml KieModule on the classpath that includes all resources in a single default KieBase. The example shows the retrieval of the default KieSession from the classpath.

An empty kmodule.xml will produce a single KieBase that includes all files found under resources path, be it DRL, BPMN2, XLS etc. That single KieBase is the default and also includes a single default KieSession. Default means they can be created without knowing their names.

**Example 4.37. Author - kmodule.xml**

<kmodule xmlns="http://jboss.org/kie/6.0.0/kmodule"> </kmodule>

**Example 4.38. Build and Install - Maven**

mvn install

ks.getKieClasspathContainer() returns the KieContainer that contains the KieBases deployed onto the environment classpath. kContainer.newKieSession() creates the default KieSession. Notice that you no longer need to look up the KieBase, in order to create the KieSession. The KieSession knows which KieBase it's associated with, and use that, which in this case is the default KieBase.

**Example 4.39. Utilize and Run - Java**

KieServices ks = KieServices.Factory.get();

KieContainer kContainer = ks.getKieClasspathContainer();

KieSession kSession = kContainer.newKieSession();

kSession.setGlobal("out", out);

kSession.insert(**new** Message("Dave", "Hello, HAL. Do you read me, HAL?"));

kSession.fireAllRules();

#### 4.2.6.2. Named KieSession

* Project: named-kiesession.
* Summary: kmodule.xml that has one named KieBase and one named KieSession. The examples shows the retrieval of the named KieSession from the classpath.

kmodule.xml will produce a single named KieBase, 'kbase1' that includes all files found under resources path, be it DRL, BPMN2, XLS etc. KieSession 'ksession1' is associated with that KieBase and can be created by name.

**Example 4.40. Author - kmodule.xml**

<kmodule xmlns="http://jboss.org/kie/6.0.0/kmodule">

<kbase name="kbase1">

<ksession name="ksession1"/>

</kbase>

</kmodule>

**Example 4.41. Build and Install - Maven**

mvn install

ks.getKieClasspathContainer() returns the KieContainer that contains the KieBases deployed onto the environment classpath. This time the KieSession uses the name 'ksession1'. You do not need to lookup the KieBase first, as it knows which KieBase 'ksession1' is assocaited with.

**Example 4.42. Utilize and Run - Java**

KieServices ks = KieServices.Factory.get();

KieContainer kContainer = ks.getKieClasspathContainer();

KieSession kSession = kContainer.newKieSession("ksession1");

kSession.setGlobal("out", out);

kSession.insert(**new** Message("Dave", "Hello, HAL. Do you read me, HAL?"));

kSession.fireAllRules();

#### 4.2.6.3. KieBase Inheritence

* Project: kiebase-inclusion.
* Summary: 'kmodule.xml' demonstrates that one KieBase can include the resources from another KieBase, from another KieModule. In this case it inherits the named KieBase from the 'name-kiesession' example. The included KieBase can be from the current KieModule or any other KieModule that is in the pom.xml dependency list.

kmodule.xml will produce a single named KieBase, 'kbase2' that includes all files found under resources path, be it DRL, BPMN2, XLS etc. Further it will include all the resources found from the KieBase 'kbase1', due to the use of the 'includes' attribute. KieSession 'ksession2' is associated with that KieBase and can be created by name.

**Example 4.43. Author - kmodule.xml**

<kbase name="kbase2" includes="kbase1">

<ksession name="ksession2"/>

</kbase>

This example requires that the previous example, 'named-kiesession', is built and installed to the local Maven repository first. Once installed it can be included as a dependency, using the standard Maven <dependencies> element.

**Example 4.44. Author - pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.drools</groupId>

<artifactId>drools-examples-api</artifactId>

<version>6.0.0/version>

</parent>

<artifactId>kiebase-inclusion</artifactId>

<name>Drools API examples - KieBase Inclusion</name>

<dependencies>

<dependency>

<groupId>org.drools</groupId>

<artifactId>drools-compiler</artifactId>

</dependency>

<dependency>

<groupId>org.drools</groupId>

<artifactId>named-kiesession</artifactId>

<version>6.0.0</version>

</dependency>

</dependencies>

</project>

Once 'named-kiesession' is built and installed this example can be built and installed as normal. Again the act of installing, will force the unit tests to run, demonstrating the use case.

**Example 4.45. Build and Install - Maven**

mvn install

ks.getKieClasspathContainer() returns the KieContainer that contains the KieBases deployed onto the environment classpath. This time the KieSession uses the name 'ksession2'. You do not need to lookup the KieBase first, as it knows which KieBase 'ksession1' is assocaited with. Notice two rules fire this time, showing that KieBase 'kbase2' has included the resources from the dependency KieBase 'kbase1'.

**Example 4.46. Utilize and Run - Java**

KieServices ks = KieServices.Factory.get();

KieContainer kContainer = ks.getKieClasspathContainer();

KieSession kSession = kContainer.newKieSession("ksession2");

kSession.setGlobal("out", out);

kSession.insert(**new** Message("Dave", "Hello, HAL. Do you read me, HAL?"));

kSession.fireAllRules();

kSession.insert(**new** Message("Dave", "Open the pod bay doors, HAL."));

kSession.fireAllRules();

#### 4.2.6.4. Multiple KieBases

* Project: 'multiple-kbases.
* Summary: Demonstrates that the 'kmodule.xml' can contain any number of KieBase or KieSession declarations. Introduces the 'packages' attribute to select the folders for the resources to be included in the KieBase.

kmodule.xml produces 6 different named KieBases. 'kbase1' includes all resources from the KieModule. The other KieBases include resources from other selected folders, via the 'packages' attribute. Note the use of wildcard '\*', to select this package and all packages below it.

**Example 4.47. Author - kmodule.xml**

<kmodule xmlns="http://jboss.org/kie/6.0.0/kmodule">

<kbase name="kbase1">

<ksession name="ksession1"/>

</kbase>

<kbase name="kbase2" packages="org.some.pkg">

<ksession name="ksession2"/>

</kbase>

<kbase name="kbase3" includes="kbase2" packages="org.some.pkg2">

<ksession name="ksession3"/>

</kbase>

<kbase name="kbase4" packages="org.some.pkg, org.other.pkg">

<ksession name="ksession4"/>

</kbase>

<kbase name="kbase5" packages="org.\*">

<ksession name="ksession5"/>

</kbase>

<kbase name="kbase6" packages="org.some.\*">

<ksession name="ksession6"/>

</kbase>

</kmodule>

**Example 4.48. Build and Install - Maven**

mvn install

Only part of the example is included below, as there is a test method per KieSession, but each one is a repetition of the other, with different list expectations.

**Example 4.49. Utilize and Run - Java**

@Test

**public** **void** **testSimpleKieBase**() {

List<Integer> list = useKieSession("ksession1");

*// no packages imported means import everything*

assertEquals(4, list.size());

assertTrue( list.containsAll( asList(0, 1, 2, 3) ) );

}

*//.. other tests for ksession2 to ksession6 here*

**private** List<Integer> **useKieSession**(String name) {

KieServices ks = KieServices.Factory.get();

KieContainer kContainer = ks.getKieClasspathContainer();

KieSession kSession = kContainer.newKieSession(name);

List<Integer> list = **new** ArrayList<Integer>();

kSession.setGlobal("list", list);

kSession.insert(1);

kSession.fireAllRules();

**return** list;

}

#### 4.2.6.5. KieContainer from KieRepository

* Project: kcontainer-from-repository
* Summary: The project does not contain a kmodule.xml, nor does the pom.xml have any dependencies for other KieModules. Instead the Java code demonstrates the loading of a dynamic KieModule from a Maven repository.

The pom.xml must include kie-ci as a depdency, to ensure Maven is available at runtime. As this uses Maven under the hood you can also use the standard Maven settings.xml file.

**Example 4.50. Author - pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.drools</groupId>

<artifactId>drools-examples-api</artifactId>

<version>6.0.0</version>

</parent>

<artifactId>kiecontainer-from-kierepo</artifactId>

<name>Drools API examples - KieContainer from KieRepo</name>

<dependencies>

<dependency>

<groupId>org.kie</groupId>

<artifactId>kie-ci</artifactId>

</dependency>

</dependencies>

</project>

**Example 4.51. Build and Install - Maven**

mvn install

In the previous examples the classpath KieContainer used. This example creates a dynamic KieContainer as specified by the ReleaseId. The ReleaseId uses Maven conventions for group id, artifact id and version. It also obeys LATEST and SNAPSHOT for versions.

**Example 4.52. Utilize and Run - Java**

KieServices ks = KieServices.Factory.get();

*// Install example1 in the local Maven repo before to do this*

KieContainer kContainer = ks.newKieContainer(ks.newReleaseId("org.drools", "named-kiesession", "6.0.0-SNAPSHOT"));

KieSession kSession = kContainer.newKieSession("ksession1");

kSession.setGlobal("out", out);

Object msg1 = createMessage(kContainer, "Dave", "Hello, HAL. Do you read me, HAL?");

kSession.insert(msg1);

kSession.fireAllRules();

#### 4.2.6.6. Default KieSession from File

* Project: default-kiesession-from-file
* Summary: Dynamic KieModules can also be loaded from any Resource location. The loaded KieModule provides default KieBase and KieSession definitions.

No kmodue.xml file exists. The project 'default-kiesession' must be built first, so that the resulting JAR, in the target folder, can be referenced as a File.

**Example 4.53. Build and Install - Maven**

mvn install

Any KieModule can be loaded from a Resource location and added to the KieRepository. Once deployed in the KieRepository it can be resolved via its ReleaseId. Note neither Maven or kie-ci are needed here. It will not set up a transitive dependency parent classloader.

**Example 4.54. Utilize and Run - Java**

KieServices ks = KieServices.Factory.get();

KieRepository kr = ks.getRepository();

KieModule kModule = kr.addKieModule(ks.getResources().newFileSystemResource(getFile("default-kiesession")));

KieContainer kContainer = ks.newKieContainer(kModule.getReleaseId());

KieSession kSession = kContainer.newKieSession();

kSession.setGlobal("out", out);

Object msg1 = createMessage(kContainer, "Dave", "Hello, HAL. Do you read me, HAL?");

kSession.insert(msg1);

kSession.fireAllRules();

#### 4.2.6.7. Named KieSession from File

* Project: named-kiesession-from-file
* Summary: Dynamic KieModules can also be loaded from any Resource location. The loaded KieModule provides named KieBase and KieSession definitions.

No kmodue.xml file exists. The project 'named-kiesession' must be built first, so that the resulting JAR, in the target folder, can be referenced as a File.

**Example 4.55. Build and Install - Maven**

mvn install

Any KieModule can be loaded from a Resource location and added to the KieRepository. Once in the KieRepository it can be resolved via its ReleaseId. Note neither Maven or kie-ci are needed here. It will not setup a transitive dependency parent classloader.

**Example 4.56. Utilize and Run - Java**

KieServices ks = KieServices.Factory.get();

KieRepository kr = ks.getRepository();

KieModule kModule = kr.addKieModule(ks.getResources().newFileSystemResource(getFile("named-kiesession")));

KieContainer kContainer = ks.newKieContainer(kModule.getReleaseId());

KieSession kSession = kContainer.newKieSession("ksession1");

kSession.setGlobal("out", out);

Object msg1 = createMessage(kContainer, "Dave", "Hello, HAL. Do you read me, HAL?");

kSession.insert(msg1);

kSession.fireAllRules();

#### 4.2.6.8. KieModule with Dependent KieModule

* Project: kie-module-form-multiple-files
* Summary: Programmatically provide the list of dependant KieModules, without using Maven to resolve anything.

No kmodue.xml file exists. The projects 'named-kiesession' and 'kiebase-include' must be built first, so that the resulting JARs, in the target folders, can be referenced as Files.

**Example 4.57. Build and Install - Maven**

mvn install

Creates two resources. One is for the main KieModule 'exRes1' the other is for the dependency 'exRes2'. Even though kie-ci is not present and thus Maven is not available to resolve the dependencies, this shows how you can manually specify the dependent KieModules, for the vararg.

**Example 4.58. Utilize and Run - Java**

KieServices ks = KieServices.Factory.get();

KieRepository kr = ks.getRepository();

Resource ex1Res = ks.getResources().newFileSystemResource(getFile("kiebase-inclusion"));

Resource ex2Res = ks.getResources().newFileSystemResource(getFile("named-kiesession"));

KieModule kModule = kr.addKieModule(ex1Res, ex2Res);

KieContainer kContainer = ks.newKieContainer(kModule.getReleaseId());

KieSession kSession = kContainer.newKieSession("ksession2");

kSession.setGlobal("out", out);

Object msg1 = createMessage(kContainer, "Dave", "Hello, HAL. Do you read me, HAL?");

kSession.insert(msg1);

kSession.fireAllRules();

Object msg2 = createMessage(kContainer, "Dave", "Open the pod bay doors, HAL.");

kSession.insert(msg2);

kSession.fireAllRules();

#### 4.2.6.9. Programmaticaly build a Simple KieModule with Defaults

* Project: kiemoduelmodel-example
* Summary: Programmaticaly buid a KieModule from just a single file. The POM and models are all defaulted. This is the quickest out of the box approach, but should not be added to a Maven repository.

**Example 4.59. Build and Install - Maven**

mvn install

This programmatically builds a KieModule. It populates the model that represents the ReleaseId and kmodule.xml, and it adds the relevant resources. A pom.xml is generated from the ReleaseId.

**Example 4.60. Utilize and Run - Java**

KieServices ks = KieServices.Factory.get();

KieRepository kr = ks.getRepository();

KieFileSystem kfs = ks.newKieFileSystem();

kfs.write("src/main/resources/org/kie/example5/HAL5.drl", getRule());

KieBuilder kb = ks.newKieBuilder(kfs);

kb.buildAll(); *// kieModule is automatically deployed to KieRepository if successfully built.*

**if** (kb.getResults().hasMessages(Level.ERROR)) {

**throw** **new** RuntimeException("Build Errors:\n" + kb.getResults().toString());

}

KieContainer kContainer = ks.newKieContainer(kr.getDefaultReleaseId());

KieSession kSession = kContainer.newKieSession();

kSession.setGlobal("out", out);

kSession.insert(**new** Message("Dave", "Hello, HAL. Do you read me, HAL?"));

kSession.fireAllRules();

#### 4.2.6.10. Programmaticaly build a KieModule using Meta Models

* Project: kiemoduelmodel-example
* Summary: Programmaticaly build a KieModule, by creating its kmodule.xml meta model resources.

**Example 4.61. Build and Install - Maven**

mvn install

This programmatically builds a KieModule. It populates the model that represents the ReleaseId and kmodule.xml, as well as add the relevant resources. A pom.xml is generated from the ReleaseId.

**Example 4.62. Utilize and Run - Java**

KieServices ks = KieServices.Factory.get();

KieFileSystem kfs = ks.newKieFileSystem();

Resource ex1Res = ks.getResources().newFileSystemResource(getFile("named-kiesession"));

Resource ex2Res = ks.getResources().newFileSystemResource(getFile("kiebase-inclusion"));

ReleaseId rid = ks.newReleaseId("org.drools", "kiemodulemodel-example", "6.0.0-SNAPSHOT");

kfs.generateAndWritePomXML(rid);

KieModuleModel kModuleModel = ks.newKieModuleModel();

kModuleModel.newKieBaseModel("kiemodulemodel")

.addInclude("kiebase1")

.addInclude("kiebase2")

.newKieSessionModel("ksession6");

kfs.writeKModuleXML(kModuleModel.toXML());

kfs.write("src/main/resources/kiemodulemodel/HAL6.drl", getRule());

KieBuilder kb = ks.newKieBuilder(kfs);

kb.setDependencies(ex1Res, ex2Res);

kb.buildAll(); *// kieModule is automatically deployed to KieRepository if successfully built.*

**if** (kb.getResults().hasMessages(Level.ERROR)) {

**throw** **new** RuntimeException("Build Errors:\n" + kb.getResults().toString());

}

KieContainer kContainer = ks.newKieContainer(rid);

KieSession kSession = kContainer.newKieSession("ksession6");

kSession.setGlobal("out", out);

Object msg1 = createMessage(kContainer, "Dave", "Hello, HAL. Do you read me, HAL?");

kSession.insert(msg1);

kSession.fireAllRules();

Object msg2 = createMessage(kContainer, "Dave", "Open the pod bay doors, HAL.");

kSession.insert(msg2);

kSession.fireAllRules();

Object msg3 = createMessage(kContainer, "Dave", "What's the problem?");

kSession.insert(msg3);

kSession.fireAllRules();

## 4.3. Security

### 4.3.1. Security Manager

The KIE engine is a platform for the modelling and execution of business behavior, using a multitude of declarative abstractions and metaphores, like rules, processes, decision tables and etc.

Many times, the authoring of these metaphores is done by third party groups, be it a different group inside the same company, a group from a partner company, or even anonymous third parties on the internet.

Rules and Processes are designed to execute arbitrary code in order to do their job, but in such cases it might be necessary to constrain what they can do. For instance, it is unlikely a rule should be allowed to create a classloader (what could open the system to an attack) and certainly it should not be allowed to make a call toSystem.exit().

The Java Platform provides a very comprehensive and well defined security framework that allows users to define policies for what a system can do. The KIE platform leverages that framework and allow application developers to define a specific policy to be applied to any execution of user provided code, be it in rules, processes, work item handlers and etc.

1. Decision Making [↑](#footnote-ref-1)
2. Spaghetti Code [↑](#footnote-ref-2)
3. Application Life-cycle [↑](#footnote-ref-3)
4. Debugging [↑](#footnote-ref-4)
5. Declarative [↑](#footnote-ref-5)
6. Business Rules [↑](#footnote-ref-6)
7. Process [↑](#footnote-ref-7)
8. Business Rule Engine (BRE) [↑](#footnote-ref-8)
9. Facts [↑](#footnote-ref-9)
10. Artificial Intelligence (A.I) [↑](#footnote-ref-10)
11. Neural Network [↑](#footnote-ref-11)
12. Genetic Algorithms [↑](#footnote-ref-12)
13. Expert Systems [↑](#footnote-ref-13)
14. Knowledge Representation [↑](#footnote-ref-14)
15. Business Rule Management Systems [↑](#footnote-ref-15)
16. Production Rules System [↑](#footnote-ref-16)
17. matching [↑](#footnote-ref-17)
18. Matching Pattern [↑](#footnote-ref-18)
19. Object Oriented [↑](#footnote-ref-19)
20. Production Memory [↑](#footnote-ref-20)
21. Facts [↑](#footnote-ref-21)
22. Working Memory [↑](#footnote-ref-22)
23. Agenda [↑](#footnote-ref-23)
24. Conflict Resolution strategy [↑](#footnote-ref-24)
25. Forward Chaining [↑](#footnote-ref-25)
26. Backward Chaining [↑](#footnote-ref-26)
27. Hybrid Chaining Systems [↑](#footnote-ref-27)
28. Data-Driven [↑](#footnote-ref-28)
29. Goal-driven [↑](#footnote-ref-29)
30. Sub-Goal [↑](#footnote-ref-30)
31. Declarative Programming [↑](#footnote-ref-31)
32. Cross-domain [↑](#footnote-ref-32)
33. Multi-domain [↑](#footnote-ref-33)
34. Scalability [↑](#footnote-ref-34)
35. Domain Specific Language [↑](#footnote-ref-35)
36. Embeded [↑](#footnote-ref-36)
37. Stateful [↑](#footnote-ref-37)
38. Procedural [↑](#footnote-ref-38)
39. Non-procedural [↑](#footnote-ref-39)
40. Interfere [↑](#footnote-ref-40)
41. Versioning [↑](#footnote-ref-41)
42. Repository [↑](#footnote-ref-42)
43. Business Rule Management System [↑](#footnote-ref-43)
44. Rule Base [↑](#footnote-ref-44)
45. Knowledge Base [↑](#footnote-ref-45)
46. Inference Engine [↑](#footnote-ref-46)
47. Declarative [↑](#footnote-ref-47)
48. Non-procedural [↑](#footnote-ref-48)
49. Rule Template [↑](#footnote-ref-49)
50. Place holder [↑](#footnote-ref-50)
51. Rule Syntax Checking [↑](#footnote-ref-51)
52. Rule flows [↑](#footnote-ref-52)
53. Decision Tables and decision Tree [↑](#footnote-ref-53)
54. Data-Driven [↑](#footnote-ref-54)
55. Goal-driven [↑](#footnote-ref-55)
56. Sub-Goal [↑](#footnote-ref-56)
57. Uncertainty [↑](#footnote-ref-57)
58. Explanation [↑](#footnote-ref-58)
59. Java Virtual Machine [↑](#footnote-ref-59)
60. Domain Specific Language [↑](#footnote-ref-60)
61. KIE : stands for " Knowledge Is Everything" [↑](#footnote-ref-61)
62. Plugins [↑](#footnote-ref-62)
63. Java Virtual Machine [↑](#footnote-ref-63)
64. Domain Specific Language [↑](#footnote-ref-64)
65. KIE : stands for " Knowledge Is Everything" [↑](#footnote-ref-65)
66. Plugins [↑](#footnote-ref-66)
67. Knowledge Is Everything [↑](#footnote-ref-67)
68. Session [↑](#footnote-ref-68)