# はじめに

Fuzzy Logic System(FLS)は，さまざまなアプリケーション分野で幅広い問題をうまく解決できるため，広く使用されています．実際，設計および開発された様々なアプリケーションとほぼ同じ数のFLSがあります．ただし，通常，次の共通の構造を有します． スクリーンショット が含まれている画像

自動的に生成された説明

この構造によれば，ファジィ化段階ではファジィでない入力をファジィ入力に変換し，非ファジィ化段階では出力に対し正反対の働きをします．ファジィ推論機構は，ファジィ入力を処理し，ファジィ出力を生成します．そのために，推論メカニズムはknowledgebase(すべての関連するファジィ変数の定義)およびrulrbase(知識ベースで定義された変数間のすべてのファジィ関係のセット)に従って与えられた入力を解釈します．

FLSの複製とアプリケーションには，高度な知識と経験が必要とされます．さらに，[Alcala-Fdez and Alonso](https://sci2s.ugr.es/fss)によって完全に説明されているように，自身の提案に関連したソフトウェアやソースコードを公開する研究者はほとんどいません．このことは学会や工業などの他の分野の科学的進歩に対する主要な障害になっています．近年では，エラーの迅速な検出，革新的なアプリケーション，FLSの速い導入などの多くの利点を提供するようFLSのオープンソースソフトウェアが開発されています．: [**IEEE ComputationalIntelligence Society**](IEEE%20ComputationalIntelligence%20Society)の[**Fuzzy Systems Technical Committee**](https://cis.ieee.org/fuzzy-systems-tc.html)で[**Task Force on Fuzzy Systems Software**](https://sci2s.ugr.es/TF-FSS)の助言によって，研究者や開発者は，新しいソフトウェアを公開するときにいくつかの重要な考慮事項(相互運用性，新規性，有用性，および関連性)について慎重に検討する必要があります．

**JFML** は，ファジィシステムの相互運用性と有用性の促進を目的としたオープンソースのJavaライブラリです．その新規性と関連性は**JFML**が[Standards Committee of the IEEE Computational Intelligence Society](https://cis.ieee.org/standards-committee.html)が支援し，the new IEEE Std 1855が公開した世界で最初のライブラリであることに起因します．

**JFML** は[GitHub](https://github.com/sotillo19/JFML.git)から自由にダウンロードできます．**JFML**を構築するためには，[git@github.com:sotillo19/JFML.git](mailto:git@github.com:sotillo19/JFML.git) から .gitリポジトリ をクローンするか， .zipファイルでダウンロードしてください．次の手順に従ってお勧めします．

[GitHub](https://github.com/sotillo19/JFML.git)からクローンまたはダウンロードを選択するか， [latest release](https://github.com/sotillo19/JFML/releases)から .zipフイァイルでライブラリをダウンロードし，ローカルフォルダにて解凍してください．

1. Eclipseにて新しいプロジェクトを作成してください．その際，前の手順で生成されたフォルダをワークスペースとして選択することを忘れないでください．
2. プロジェクトの依存関係を生成するために buildJFML.xml を Ant Build として実行してください．そのためにxmlファイルを右クリックし適切なオプションを選択してください．ant は自由に使用可能で それは通常Eclipseと統合されています．
3. ソースファイルを変更する場合は，ソースファイルをコンパイルし，jar ファイルを生成するために buildJFML.xml をAnt Build として実行してください．

ソースコードのzipファイルには，Examples フォルダで使用可能なコンパイル済みのjarライブラリが含まれていることに注意してください．これは buildJAR.xmlの実行後に上書きされます．

**Py4JFMLは** の**JFML**ためのPythonラッパーで，[GitHub](https://github.com/cmencar/py4jfml)から自由にからダウンロードすることができます．**Py4JFML**を構築するには，git@github.com:cmencar/PyJFML.git から .gitリポジトリ をクローンするか， .zipファイルでダウンロードしてください．

# IEEE Standard 1855-2016 - Fuzzy Markup Language (FML)

Fuzzy Markup Language(FML)という新しい規格言語が[IEEEStandard 1855-2016](https://standards.ieee.org/standard/1855-2016.html)で提案され，可読性のあるハードウェアに依存しない方法でファジィ論理システムをモデル化するため，eXtensible Markup Language(XML)仕様と関連ツールによって得られる利点を開発しています．したがって，産業用ファジィシステムの設計者には統一された高レベルの説明された相互運用可能なファジィシステムの方法論が提供されます．The W3C XML Schema definition languageは，FMLプログラムの構文と意味論を定義する基準として用いられています．

# Java API - JFML

IEEE規格の概要に含まれる4種類のファジィシステムを簡単に作成できます．システムの名前で FuzzyInferenceSystemオブジェクト を作成する必要があります．次に，knowledgebaseの定義とrulebaseの定義を追加します．

次のJavaコードは，「tipper」という名前のFuzzyInferenceSystemオブジェクトを作成します．

FuzzyInferenceSystem tipper = new FuzzyInferenceSystem( "tipper");

次のサブセクションでは，knowledgebaseとrulebaseを作成する方法を示します．

# Knowledge Base

最初に，システム変数の定義を含むKnowledgeBaseTypeオブジェクトを作成し，それをファジィシステムに追加する必要があります．

KnowledgeBaseType kb = new KnowledgeBaseType();

tipper.setKnowledgeBase(kb);

次に，変数の定義がknowledgebaseに含まれます． 5つのタイプの変数を定義できます

### FuzzyVariableType:

入力または出力のファジィ変数を表します．このクラスの各オブジェクトは次の情報を持ちます．

* + *Name*:ファジィ変数の一意の名前を持つstringです．この情報は変数を作成するときに必要です．
  + *scale*:ファジィ変数の測定に使用されるスケールを持つstringです．変数を作成するとき，この情報は任意です．
  + *DomainLeft*:ファジィ変数のdiscourseの領域の左境界を表すfloatです．この情報は，変数を作成するときに必要です．
  + *DomainRight*:ファジィ変数のdiscourseの領域の右境界を表すfloatです．この情報は，変数を作成するときに必要です．
  + *Type*:ルール内のファジィ変数の位置(後続部分または先行部分)を持つstringです．その値は” input” または ”output” です．変数を生成するとき，この情報は任意であり，値を指定しない場合，デフォルト値は ”input” となります．
  + *Accumulation*:これは，この変数がルールの後続部分に関係している場合に使用される累積方法のstringです．変数を作成するとき，この情報は任意です．デフォルト値はMAXですが，次の累積方法のいずれかを選択できます
    - * MAX:最大
      * PROBOR:確率的合計
      * BSUM:有限界合計
      * DRS:動的合計
      * ESUM:アインシュタイン合計
      * HSUM:ハマッチャー合計
      * NILMAX:べき乗の最大値
      * custom\_ \ S \*:ユーザーが実装するカスタム累計方法
  + *Defuzzifier*:これは，この変数がルールの後続部分に関係している場合に使用される非ファジィ化フィルターを持つstringです．変数を作成するとき，この情報は任意です．デフォルト値はCOGですが，次のいずれかを選択できます:
    - * COG:重心
      * MOM:最大値の平均
      * LM:左端の最大
      * RM:右端の最大
      * COA:領域の中心
      * custom\_ \ S \*:ユーザー が実装するカスタム非ファジィ化フィルター
  + *DefaultValue*:これは，問題の変数に対してルールが実行されていない場合にのみ使用されるfloatです．変数を作成するとき，この情報は任意です．デフォルト値は0です ．
  + *NetworkAddress*:コンピュータネットワーク内のファジィ変数の場所を示すstringです．変数を作成するとき，この情報は任意です．デフォルト値は "127.0.0.1" です．

次のJavaコードは，左境界0，右境界10，および残りの属性がデフォルト値の ”food” という名前の入力ファジィ変数を作成します．さらに，この変数はファジィシステムのknowledgebaseに追加されます .

FuzzyVariableType food = new FuzzyVariableType("food"，0，10);

kb.addVariable(food);

次に，変数の言語の項を定義します．各項には，属性名(string)，補数(string "true"または "false")，メンバーシップ関数のタイプ，および対応するファジィセットに必要なパラメータを持つfloat listが含まれます．属性補完は任意であり，そのデフォルト値は ”false” であることに注意してください． JFMLで次のメンバーシップ関数を定義できます:

1. Triangular:このオプションは，3つのパラメータa，b，cに応じて三角関数を定義します．次のJavaコードは， a = 0，b = 2，c = 4 で ”rancid” という名前の関数の項を生成し，変数 “food” に追加します.

FuzzyTermType rancid = new FuzzyTermType( "rancid"，FuzzyTermType．TYPE\_triangularShape，(new float [] {0f，2f，4f}));

food.addFuzzyTerm(rancid);

1. Left Linear:このオプションは，2つのパラメータa，bに応じて左辺直線型関数を定義します．次のJavaコードは， a = 0，b = 4 で ”rancid” という名前の関数の項を生成し，変数 “food” に追加します.

FuzzyTermType rancid = new FuzzyTermType( "rancid"，FuzzyTermType.TYPE\_leftLinearShape，(new float [] {0f，4f}));

food.addFuzzyTerm(rancid);

1. Right Linear:このオプションは，2つのパラメータa，bに応じて右辺直線型関数を定義します．次のJavaコードは， a = 4，b = 10 で ”delicious” という名前の関数の項を生成し，変数 “food” に追加します.

FuzzyTermType delicious = new FuzzyTermType( "delicious"，FuzzyTermType.TYPE\_rightLinearShape，(new float [] {4f，10f}));

food.addFuzzyTerm(delicious);

1. Trapezoidal:このオプションは，a，b，c，dの4つのパラメータに応じて台形関数を定義します．次のJavaコードは， a = 4，b = 10 で ”rancid” という名前の関数の項を生成し，変数 “food” に追加します.

FuzzyTermType rancid = new FuzzyTermType( "rancid "，FuzzyTermType.TYPE\_trapezoidShape，(new float [] {0f，1f，3f，4f}));

food.addFuzzyTerm(rancid);

1. Rectangular Shape:このオプションは，2つのパラメータa，bに応じて長方形関数を定義します．次のJavaコードは， a = 4，b = 7 で ”good” という名前の関数の項を生成し，変数 “food” に追加します.

FuzzyTermType good = new FuzzyTermType( "good"，FuzzyTermType.TYPE\_rectangularShape，(new float [] {4f，7f}));

food.addFuzzyTerm(good);

1. Gaussi:このオプションは，2つのパラメータc，σに応じて対称ガウス関数を定義します．次のJavaコードは， a = 5，σ= 1 で ”good” という名前の関数の項を生成し，変数 “food” に追加します.

FuzzyTermType good = new FuzzyTermType( "good"，FuzzyTermType.TYPE\_gaussianShape，(new float [] {5f，1f}));

food.addFuzzyTerm(good);

1. Left Gaussian:このオプションは，2つのパラメータc，σに応じて左ガウス関数を定義します．次のJavaコードは， a = 2，σ= 1 で ”rancid” という名前の関数の項を生成し，変数 “food” に追加します.

FuzzyTermType rancid = new FuzzyTermType( "rancid"，FuzzyTermType.TYPE\_leftGaussianShape，(new float [] {2f，1f}));

food.addFuzzyTerm(rancid);

1. Right Gaussian:このオプションは，2つのパラメータc，σに応じて左ガウス関数を定義します．次のJavaコードは， a = 8，σ= 1 で ”delicius” という名前の関数の項を生成し，変数 “food” に追加します.

FuzzyTermType delicious = new FuzzyTermType( "delicious"，FuzzyTermType.TYPE\_rightGaussianShape，(new float [] {8f，1f}));

food.addFuzzyTerm(delicious);

1. Pi-Shaped:このspline-basedカーブは，そのΠ型からその名が付けられ，4つのパラメータa，b，c，およびdに依存します．次のJavaコードは， a = 2，b = 4，c = 6，d = 8 で ”delicius” という名前の関数の項を生成し，変数 “food” に追加します.

FuzzyTermType good = new FuzzyTermType( "good "，FuzzyTermType.TYPE\_piShape，(new float [] {2f，4f，6f，8f}));

food.addFuzzyTerm(good);

1. Z-Shape:このspline-basedカーブは，そのZ型からその名が付けられ，2つのパラメータaとbに依存します．次のJavaコードは， a = 2，b = 4 で ”rancid” という名前の関数の項を生成し，変数 “food” に追加します.

FuzzyTermType rancid = new FuzzyTermType( "rancid"，FuzzyTermType.TYPE\_zShape，(new float [] {2f，4f}));

food.addFuzzyTerm(rancid);

1. S-Shape:このspline-based カーブは，そのS型からその名が付けられ，2つのパラメータaとbに依存しています．次のJavaコードは， a = 2，b = 4 で ” delicious” という名前の関数の項を生成し，変数 “food” に追加します.

FuzzyTermType delicious = new FuzzyTermType( "delicious"，FuzzyTermType.TYPE\_sShape，(new float [] {6f，8f}));

food.addFuzzyTerm(delicious);

1. Singleton Shape:このオプションは， 1つのパラメータに応じてシングルトン関数を定義します． 次のJavaコードは， a = 5 で ” good” という名前の関数の項を生成し，変数 “food” に追加します．

FuzzyTermType good = new FuzzyTermType( "good"，FuzzyTermType.TYPE\_singletonShape，(new float [] { 5f}));

food.addFuzzyTerm(good);

1. PointSet Shape:このオプションは，ユーザーによって定義されたポイントのセットによって関数を定義します(区間関数)．この関数は，2つの追加の属性(interpolationMethod and degree)を持ち，パラメータのリストは，ユーザーによって定義されたpointType オブジェクトのlist である.

* interpolationMethod:これは，ユーザーによって定義されたポイントを補間する方法を定義します．この属性は，次の値をテーブル化できます．"linear" : 線形補間を実装します．"lagrange" : ラグランジュ補間を実装します．"spline” スプライン補間を実装します．この属性は任意であり，そのデフォルト値は "linear" です．
* Degree:多項式の次数は，ユーザーが "lagrange" 又は "spline” を選択した時，補間処理の際に使用されます．

次のJavaコードは，”delicious” という名前の関数の項を生成し，補間手法はデフォルトでlistのポイントは(5, 0), (8, 0.5), (9, 1)です．

List points = new ArrayList();

points.add(new PointType(5f, 0f));

points.add(new PointType(8f, 0.5f));

points.add(new PointType(9f,1f));

FuzzyTermType delicious = new FuzzyTermType("delicious",

FuzzyTermType.TYPE\_pointSetShape, points);

food.addFuzzyTerm(delicious);

1. Circular Definition:このオプションでは，あらかじめ定義されたファジィの項を含む論理的表現の手法によって新しい項を定義することができます． このオプションでは，パラメータのlistは循環定義(CircularDefinitionType):(これは次の論理要素から選択された1つの要素を含みます．)あるいは(AndLogicalTypeオブジェクト)または(OrLogicalTypeオブジェクト)． AndLogicalTypeオブジェクトには，3つの属性があります． t-normの名前，変数の項，またはLogicalTypeオブジェクト(AndLogicalTypeまたはOrLogicalTypeの場合があります) の名前，そして変数の項の別の名前またはLogicalTypeオブジェクト(AndLogicalTypeまたはOrLogicalTypeの場合があります)です． OrLogicalTypeオブジェクトには3つの属性があります．t-conormの名前．変数の項またはLogicalTypeオブジェクト(AndLogicalTypeまたはOrLogicalTypeの場合があります)の名前，変数の項の別の名前またはLogicalTypeオブジェクト (AndLogicalTypeまたはOrLogicalTypeの場合があります)です．

使用できるt-norms(デフォルト値はMIN):

* + - MIN:オペレーターの最小値
    - PROD:オペレーターの積
    - BDIF:有界差
    - DRP:動的積
    - EPROD:Einstein積
    - HPROD:Hamacher積
    - NILMIN: べき乗の最小値
    - custom\_\S\*: カスタム手法

使用できるt- conorms(デフォルト値はMIN):

* + - MAX:オペレーターの最大
    - PROBOR:確率的合計
    - BSUM:有限界合計
    - DRS:動的合計
    - ESUM:アインシュタイン合計
    - HSUM:ハマッチャー合計
    - NILMAX:べき乗の最大値
    - custom\_ \ S \*:カスタム手法

次のJavaコードは，”correct” という名前の関数の項を生成し，OrLogicalTypeオブジェクトをt- conorm MAX と項 ”good” (pi-Shaped で生成された)と項 “delicious” (s-Shaped で生成された)で使用し．変数 “food” の新しい項の循環定義を生成します．

CircularDefinitionType c = new CircularDefinitionType (new OrLogicalType ("MAX", "good", "delicious"), food);

FuzzyTermType correct = new FuzzyTermType("correct",FuzzyTermType.TYPE\_circularDefinition, c);

food.addFuzzyTerm(correct);

1. Custom Membership Function: ユーザーによって定義されたカスタム関数．

### AggregatedFuzzyVariableType:

これはシステムに関連する2つ以上の変数を集計するために使用できる入力ファジィ変数を表します．このクラスの変数には次の3つの属性が含まれています．

* *Name*:ファジィ変数の一意の名前を持つstringです．この情報は，変数を作成するときに必要です．
* *Type*:ルール内のファジィ変数の位置を含むstringです．このタイプの変数では，この属性は常に ”input” です．
* *NetworkAddress*:コンピュータネットワーク内のファジィ変数の場所を示すstringです．変数を作成するとき，この情報は任意です．デフォルト値は "127.0.0.1"です．

次のJavaコードは，"aggregated" という名前の入力ファジィ変数を作成します．さらに，この変数はファジィシステムのknowledgebaseに追加されます

AggregatedFuzzyVariableType aggregated = new AggregatedFuzzyVariableType( "aggregated"， "input");

kb.addVariable(aggregated);

各変数には，変数の項を含むAggregatedFuzzyTermオブジェクトのlistも持ちます．各AggregatedFuzzyTermオブジェクトは，項の名前を持つ属性と，(AndAggregatedType)または(OrAggregatedType)の論理オブジェクトを持ちます．各AndAggregatedTypeまたはOrAggregatedTypeオブジェクトには，それぞれ属性t-norm(循環定義と同じように使用できます)とt-conorn(循環定義と同じように使用できます)のみが含まれます．さらに，どちらのオブジェクトにも次の2つの要素のいずれかが含まれています

* 2つのClauseTypeオブジェクト．
* AndAggregatedTypeオブジェクトとClauseTypeオブジェクト．
* OrAggregatedTypeオブジェクトとClauseTypeオブジェクト．

各ClauseTypeオブジェクトには3つの要素が含まれます．

Modifier: 使用した修飾子のstringです．この情報は任意であるが，我々は，これらの修飾子のいずれかを選択することができる．

* + - above : 次元の最大値によって特徴付けられた項で，xmaxと名付けられたx軸上の最初の点より上の領域を識別する境界
    - any:任意境界．
    - below : 次元の最大値によって特徴付けられた項で，xmaxと名付けられたx軸上の最初の点より下の領域を識別する境界
    - extremely : 極端境界
    - intensify : 集約境界
    - more\_or\_less : 多かれ少なかれ境界
* norm : 関数の最大メンバーシップ次元のxmaxがあるnormの境界．
* not : 境界無し．
* plus : 追加境界．
* seldom : seldom境界
* slightly : slightly境界
* Somewhat : somewhat境界
* very : very境界
* custom : カスタム境界

このタイプの変数を説明するために， 3つの項(low, middle, high)を持つ2つのファジィ変数(food, service)があるknowledgebaseを考えてみましょう．新しい変数 “aggregated" の次の項を作成できます

AggregatedFuzzyTermType term1 = new AggregatedFuzzyTermType( "bad"，new AndAggregatedType(new ClauseType(food，(FuzzyTerm)food.getTerm( "low"))，new ClauseType(service， (FuzzyTerm)food.getTerm( "low")))));

AggregatedFuzzyTermType term2 = new AggregatedFuzzyTermType( "good"，new AndAggregatedType(new ClauseType(food，(FuzzyTerm)food.getTerm( "high"))，new ClauseType(service，(FuzzyTerm)food.getTerm( "high"))))) ;

aggregated.addAggregatedFuzzyTerm(term1);

aggregated.addAggregatedFuzzyTerm(term2);

### TsukamotoTermType:

tukamotoシステムのruleaseで使用できる出力ファジィ変数を表します．このクラスの各オブジェクトには，次の情報が含まれています．

* Name : ファジィ変数の一意の名前を持つstringです．この情報は，変数を作成するときに必要です．
* *scale*:ファジィ変数の測定に使用されるスケールを持つstringです．変数を作成するとき，この情報は任意です．
* *DomainLeft*:ファジィ変数のdiscourseの領域の左境界を表すfloatです．この情報は，変数を作成するときに必要です．
* *DomainRight*:ファジィ変数のdiscourseの領域の右境界を表すfloatです．この情報は，変数を作成するときに必要です．
* *Type*:ルール内のファジィ変数の位置(後続部分または先行部分)を持つstringです．その値は ”output” である必要があります．変数を生成するとき，この情報は任意であり，値を指定しない場合，デフォルト値は ”output” となります．
* Combination : これは，変数がルールの後続部分に関係するときに使われる集約手法のstringです．変数を作成するとき，この情報は任意です．デフォルト値は “WA” ですが，次の集計方法のいずれかを選択できます ．
  + WA:加重平均
  + custom\_ \ S \*:カスタム集計方法
* *DefaultValue*:これは，問題の変数に対してルールが実行されていない場合にのみ使用されるfloatです．変数を作成するとき，この情報は任意です．デフォルト値は0です ．
* *NetworkAddress*:コンピュータネットワーク内のファジィ変数の場所を示すstringです．変数を作成するとき，この情報は任意です．デフォルト値は "127.0.0.1" です．

次のJavaコードは，左境界0，右境界20，および残りの属性のデフォルト値の ”tip” という名前の入力ファジィ変数を作成します．さらに，この変数はファジィシステムのknowldgebaseに追加されます．

TsukamotoVariableTypetip = new TsukamotoVariableType( "tip"，0，20);

kb.addVariable(tip);

次に，変数の単調な言語的項を定義します．各項は，name(dtring)，complement(string ”yes” または “no” )，メンバーシップ関数のタイプ，および対応するファジィセットに必要なパラメータを含むfloat listなどの属性を持ちます．Attribute comlementは任意であり，そのデフォルト値は ”default” です．次のモノトーンメンバーシップ関数を使用できます．

* Left Linear:このオプションは，2つのパラメータa，bに応じて左辺直線型関数を定義します．次のJavaコードは， a = 0，b = 10 で”cheap” という名前の関数の項を生成し，変数 “tip” に追加します.

TsukamotoTermType cheap = new TsukamotoTermType ("cheap",FuzzyTermType.TYPE\_leftLinearShape,(new float[] {0f,10f}));

tip.addTsukamotoTerm(cheap);

* Right Linear:このオプションは，2つのパラメータa，bに応じて左辺直線型関数を定義します．次のJavaコードは， a = 15，b = 20 で”generous” という名前の関数の項を生成し，変数 “tip” に追加します.

TsukamotoTermType generous = new TsukamotoTermType ("generous",FuzzyTermType.TYPE\_rightLinearShape,(new float[] {15f,20f}));

tip.addTsukamotoTerm(generous);

* Left Gaussian:このオプションは，2つのパラメータc，σに応じて左ガウス関数を定義します．次のJavaコードは， c = 5，σ= 2 で ”cheap” という名前の関数の項を生成し，変数 “tip” に追加します.

TsukamotoTermType cheap = new TsukamotoTermType ("cheap",FuzzyTermType.TYPE\_leftGaussianShape,(new float[] {5f,2f}));

tip.addTsukamotoTerm(cheap);

* Right Gaussian:このオプションは，2つのパラメータc，σに応じて左ガウス関数を定義します．次のJavaコードは， a = 18，σ= 2 で ”generous” という名前の関数の項を生成し，変数 “tip” に追加します.

TsukamotoTermType generous = newTsukamotoTermType ("generous",FuzzyTermType.TYPE\_rightGaussianShape,(new float[] {18f,2f}));

tip.addTsukamotoTerm(generous);

* Z-Shape:このspline-basedカーブは，そのZ型からその名が付けられ，2つのパラメータaとbに依存します．次のJavaコードは， a = 5，b = 10 で ”cheap” という名前の関数の項を生成し，変数 “tip” に追加します.

TsukamotoTermType cheap = new TsukamotoTermType ("cheap",

FuzzyTermType.TYPE\_zShape,(new float[] {5f, 10f}));

tip.addTsukamotoTerm(cheap);

* S-Shape:このspline-based カーブは，そのS型からその名が付けられ，2つのパラメータaとbに依存しています．次のJavaコードは， a = 5，b = 18 で ”generous” という名前の関数の項を生成し，変数 “tip” に追加します.

TsukamotoTermType generous = new TsukamotoTermType("generous",FuzzyTermType.TYPE\_sShape, (new float[] {15f, 18f}));

tip.addTsukamotoTerm(generous);

* PointSet Shape:このオプションは，ユーザーによって定義されたポイントのセットによって関数を定義します(単調区間関数)．この関数は，追加の属性(interpolationMethod)を持ち，パラメータのリストは，ユーザーによって定義されたpointType オブジェクトのlist である.
  + - interpolationMethod:これは，ユーザーによって定義されたポイントを補間する方法を定義します．この属性は，次の値をテーブル化できます．"linear" : 線形補間を実装します．"cubic" : 単調立法補間を実装します．この属性は任意であり，そのデフォルト値は "linear" です．

次のJavaコードは，”generous” という名前の関数の項を生成し，補間手法はデフォルトでlistのポイントは(15, 0), (17, 0.5), (19, 1)です．

List points = new ArrayList();

points.add(new PointType(15f, 0f));

points.add(new PointType(17f, 0.5f));

points.add(new PointType(19f, 1f));

TsukamotoTermType generous = new

TsukamotoTermType("generous",

FuzzyTermType.TYPE\_pointSetMonotonicShape, points);

tip.addTsukamotoTerm(generous);

* Custom Membership Function : ユーザーによって実装されたカスタム単調関数

### TskVariableType:

TSKシステムのruleaseで使用できる出力ファジィ変数を表します．このクラスの各オブジェクトには，次の情報が含まれています．

* + - Name : ファジィ変数の一意の名前を持つstringです．この情報は，変数を作成するときに必要です．
    - *scale*:ファジィ変数の測定に使用されるスケールを持つstringです．変数を作成するとき，この情報は任意です．
    - *Type*:ルール内のファジィ変数の位置(後続部分または先行部分)を持つstringです．その値は ”output” である必要があります．変数を生成するとき，この情報は任意であり，値を指定しない場合，デフォルト値は ”output” となります．
    - Combination : これは，変数がルールの後続部分に関係するときに使われる集約手法のstringです．変数を作成するとき，この情報は任意です．デフォルト値は “WA” ですが，次の集計方法のいずれかを選択できます ．
    - WA:加重平均
    - custom\_ \ S \*:カスタム集計方法
    - *DefaultValue*:これは，問題の変数に対してルールが実行されていない場合にのみ使用されるfloatです．変数を作成するとき，この情報は任意です．デフォルト値は0です ．
    - *NetworkAddress*:コンピュータネットワーク内のファジィ変数の場所を示すstringです．変数を作成するとき，この情報は任意です．デフォルト値は "127.0.0.1" です．

次のJavaコードは，左境界0，右境界20，および残りの属性のデフォルト値の ”tip” という名前のtsk変数を作成します．さらに，この変数はファジィシステムのknowledgebaseに追加されます

TskVariableType tip = new TskVariableType( "tip"，0，20);

kb.addVariable(tip);

次に，変数の言語的項を定義します．各項は，name(dtring)，order(string ”yes” または “no” )，多項式に必要なパラメータを含むfloat listなどの属性を持ちます．属性はTSKシステムの順序の説明を必要とします．orderが1のとき，listの最初の値は多項式の項に依存し，残りの重みはknowledgebaseの変数のorderに従って割り当てられることに気を付けてください．簡単な例のために2つの入力ファジィ変数(food, service)があるknowledgebaseを考えてみましょう．次の項をorder 0と1と定義できます．

TskTermType average = new TskTermType( "average"，0，(new float [] {10.5f})); // y = 10.5

TskTermType cheap = new TskTermType( "cheap"，1，(new float [] {1.5f，5.6f，6.0f})); // y = 1.5 + 5.6 \*food+ 6.0 \*service

tip.addTskTerm(average);

tip.addTskTerm(cheap);

### AnYaDataCloudType :

AnYaシステムのruleaseの先行部分で使用できるdata cloudを表します．このクラスの各オブジェクトには，次の情報が含まれています．

* + - Name : ファジィ変数の一意の名前を持つstringです．この情報は，変数を作成するときに必要です．
    - *NetworkAddress*:コンピュータネットワーク内のファジィ変数の場所を示すstringです．変数を作成するとき，この情報は任意です．デフォルト値は "127.0.0.1" です．

次のJavaコードは，NetworkAddress属性のデフォルト値の ” foodRancidServicePoor” という名前のAnYa変数を作成します．さらに，この変数はファジィシステムのknowledgebaseに追加されます ．

AnYaDataCloudType foodRancidServicePoor = new

AnYaDataCloudType("foodRancidServicePoor");

kb.addVariable(foodRancidServicePoor);

このクラスの各オブジェクトには，クラウド内のデータを表すfloatのリストも含まれています．次のJavaコードは，データムのリストを変数 "food"に追加します

foodRancidServicePoor.setTerms((new float[] { 0.5f, 1.0f }));

# Rulebase

MamdaniRuleBase，TukamotoRuleBase，TskRuleBase，AnYaRuleBaseクラスのオブジェクトは，それぞれ，Mamdani，Tsukamoto，TSK，およびanYaファジィシステムのファジィルールセットを表します． FuzzyInferenceSystemオブジェクトは，システムのさまざまな動作を記述するために，1つまたは複数のMamdaniRuleBase，TsukamotoRulBase，TskRuleBase，AnYaRuleBaseオブジェクトを持ち，FuzzyInferenceSystemオブジェクトに追加された順序で評価されます．したがって、ファジィシステムに組み込みたいMamdaniRuleBaseType、TsukamotoRuleBaseType、TskRuleBaseType、およびAnYaRuleBaseType オブジェクトを作成し、ファジーシステムに追加する必要があります

MamdaniRuleBaseType rbMam = new

MamdaniRuleBaseType("MamdaniRB1");

tipper.addRuleBase(rbMam);

TsukamotoRuleBaseType rbTsu = new

TsukamotoRuleBaseType("TsukamotoRB1");

tipper.addRuleBase(rbTsu);

TskRuleBaseType rbTsk = new TskRuleBaseType("TSKRB1");

tipper.addRuleBase(rbTsk);

AnYaRuleBaseType rbAnY = new AnYaRuleBaseType("AnYaRB1");

tipper.addRuleBase(rbAnY);

mamdaniRuleBase、tsukamotoRuleBase、tskRuleBaseオブジェクトは5つの属性を持ちます。

name：rulebseの一意の名前を持つstringです。この情報は、rulebaseを作成するときに必要です。

activationMethod：使用される含意メソッドのstringです。この情報は任意であり、デフォルト値はMINですが、これらの含意のいずれかの方法を選択することができます．

* MIN:オペレーターの最小値
* PROD:オペレーターの積
* BDIF:有界差
* DRP:動的積
* EPROD:Einstein積
* HPROD:Hamacher積
* NILMIN: べき乗の最小値
* custom\_\S\*: カスタム手法

andMethod：rulebaseに含まれる全てのルールで使用されるandアルゴリズムを示すstringです。この情報は任意であり、デフォルト値はMINですが、これらのいずれかの方法を選択することができます．

* MIN:オペレーターの最小値
* PROD:オペレーターの積
* BDIF:有界差
* DRP:動的積
* EPROD:Einstein積
* HPROD:Hamacher積
* NILMIN: べき乗の最小値
* custom\_\S\*: カスタム手法

orMethod：rulebaseに含まれる全てのルールで使用されるorアルゴリズムを示すstringです。この情報は任意であり、デフォルト値はMAXですが、これらのいずれかの方法を選択することができます．

* + - * MAX:最大
      * PROBOR:確率的合計
      * BSUM:有限界合計
      * DRS:動的合計
      * ESUM:アインシュタイン合計
      * HSUM:ハマッチャー合計
      * NILMAX:べき乗の最大値
      * custom\_ \ S \*:ユーザーが実装するカスタム累計方法

*NetworkAddress*:コンピュータネットワーク内のファジィ変数の場所を示すstringです．変数を作成するとき，この情報は任意です．デフォルト値は "127.0.0.1" です．

MamdaniRuleBaseおよびTsukamotoRuleBaseオブジェクトには、FuzzyRuleTypeオブジェクトのlistが含まれています。 tskRuleBaseオブジェクトには、TskFuzzyRuleTypeオブジェクトのlistが含まれています。 AnYaRuleBaseオブジェクトには、AnYaRuleTypeオブジェクトのlistが含まれています。以下では、各タイプのルールについて説明します。

### FuzzyRuleType

TskFuzzyRuleTypeオブジェクトは、IF-THEN-ELSEの形式でMamdaniおよびTsukamotoルールを表します。各ルールには6つの属性があります。

* Name : ルールの一意の名前を持つstringです．この情報はルールを生成するとき必要です．
* Connector : 先行部分に関連するclausesに接続する論理オペレーターを示すstringです．デフォルト値はandでANDまたはORから選択できます．
* andMethod：コネクタにandが選択された場合にルールに使用されるandアルゴリズムを示すstringです．この情報は任意であり、MamdaniRuleBaseオブジェクトまたはTsukamotoRuleBaseオブジェクトを定義したときデフォルト値はMINですが、これらのいずれかの方法を選択することができます．
  + MIN:オペレーターの最小値
  + PROD:オペレーターの積
  + BDIF:有界差
  + DRP:動的積
  + EPROD:Einstein積
  + HPROD:Hamacher積
  + NILMIN: べき乗の最小値
  + custom\_\S\*: カスタム手法
* orMethod：コネクタにorが選択された場合にルールに使用されるorアルゴリズムを示すstringです．この情報は任意であり、MamdaniRuleBaseオブジェクトまたはTsukamotoRuleBaseオブジェクトを定義したときデフォルト値はMAXですが、これらのいずれかの方法を選択することができます．
  + - MAX:最大
    - PROBOR:確率的合計
    - BSUM:有限界合計
    - DRS:動的合計
    - ESUM:アインシュタイン合計
    - HSUM:ハマッチャー合計
    - NILMAX:べき乗の最大値
    - custom\_ \ S \*:ユーザーが実装するカスタム累計方法
* *weight：* 推論エンジンで使用されるルールの重要性を表すfloatです。この情報はオプションであり、その領域は[0、1]で、デフォルト値は1.0です。
* *NetworkAddress*：コンピュータネットワーク内のルールの場所を示すstringです。ルールを作成するとき、この情報は任意です。デフォルト値は "127.0.0.1"です。

例えば，簡単なルールの定義は以下のようになります．

FuzzyRuleType rule1 = new FuzzyRuleType("rule1", "or", "MAX", 1.0f);

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

各FuzzyRuleTypeオブジェクトは、1つのAntecedentTypeオブジェクトと1つのConsequentTypeオブジェクトで構成されています。 AntecedentTypeオブジェクトには、先行部分に含まれるMamdani条項を表すClauseTypeオブジェクトのlistが含まれています。 ConsequentTypeオブジェクトには、”then”の部分のClause Typeオジェクトと、”else”部分の任意の1つのClause Typeオブジェクトが含まれます。各ClauseTypeオブジェクトは、属性修飾子、FuzzyVariableTypeオブジェクト、およびFuzzyTermTypeオブジェクトを持ちます。Tsukamotoルールは、ルールの後続部分のClauseTypeオブジェクトの定義に単調変数のみを含むことができることに注意してください。 ClauseTypeの属性修飾子は、条項で使用される言語的項の変更を記述します。この情報は任意であり、これらの修飾子のいずれかを選択できます

* above
* any
* below
* extremely
* intensify
* more\_or\_less
* norm
* not
* plus
* seldom
* slightly
* somewhat
* very
* custom\_\S\* for a custom hedge.

したがって、次のようにルールの先行部分と後続部分を定義できます ．

FuzzyRuleType rule1 = new FuzzyRuleType("rule1", "or", "MAX",1.0f);

AntecedentType ant1 = new AntecedentType();

ant1.addClause(new ClauseType(food, rancid));

ant1.addClause(new ClauseType(service, poor, "very"));

rule1.setAntecedent(ant1);

ConsequentType con1 = new ConsequentType();

con1.addThenClause(tip, cheap);

rule1.setConsequent(con1);

rbMam.addRule(rule1);

### TskFuzzyRuleType

FuzzyRuleTypeオブジェクトは、IF-THEN-ELSEの形式でTSKルールを表します。各ルールには6つの属性があります。

* Name : ルールの一意の名前を持つstringです．この情報はルールを生成するとき必要です．
* Connector : 先行部分に関連するclausesに接続する論理オペレーターを示すstringです．デフォルト値はandでANDまたはORから選択できます．
* andMethod：コネクタにandが選択された場合にルールに使用されるandアルゴリズムを示すstringです．この情報は任意であり、MamdaniRuleBaseオブジェクトまたはTsukamotoRuleBaseオブジェクトを定義したときデフォルト値はMINですが、これらのいずれかの方法を選択することができます．
  + MIN:オペレーターの最小値
  + PROD:オペレーターの積
  + BDIF:有界差
  + DRP:動的積
  + EPROD:Einstein積
  + HPROD:Hamacher積
  + NILMIN: べき乗の最小値
  + custom\_\S\*: カスタム手法
* orMethod：コネクタにorが選択された場合にルールに使用されるorアルゴリズムを示すstringです．この情報は任意であり、MamdaniRuleBaseオブジェクトまたはTsukamotoRuleBaseオブジェクトを定義したときデフォルト値はMINですが、これらのいずれかの方法を選択することができます．
  + - MAX:最大
    - PROBOR:確率的合計
    - BSUM:有限界合計
    - DRS:動的合計
    - ESUM:アインシュタイン合計
    - HSUM:ハマッチャー合計
    - NILMAX:べき乗の最大値
    - custom\_ \ S \*:ユーザーが実装するカスタム累計方法
* *weight：* 推論エンジンで使用されるルールの重要性を表すfloatです。この情報はオプションであり、その領域は[0、1]で、デフォルト値は1.0です。
* *NetworkAddress*：コンピュータネットワーク内のルールの場所を示すstringです。ルールを作成するとき、この情報は任意です。デフォルト値は "127.0.0.1"です。

例えば，簡単なルールの定義は以下のようになります．

TskFuzzyRuleType rule1 = new TskFuzzyRuleType("rule1", "or","MAX", 1.0f);

各TskFuzzyRuleTypeオブジェクトは、1つのAntecedentTypeオブジェクトと1つのTskConsequentTypeオブジェクトで構成されています。 AntecedentTypeオブジェクトには、先行部分に含まれるMamdani条項を表すClauseTypeオブジェクトのlistが含まれています。TskConsequentTypeオブジェクトは、TSKシステム（定数または線形関数）の”then”ルール部分を表す（ルールのelse部分を使用する場合はもう1つ）TskClauseTypeオジェクトを持ちます．各ClauseTypeオブジェクトは、属性修飾子、FuzzyVariableTypeオブジェクト、およびFuzzyTermTypeオブジェクトを持ちます。各TskClauseTypeオブジェクトには、TskVariableTypeオブジェクトとTskTermTypeオブジェクトがあります。

したがって、次のようにルールの先行部分と後続部分を定義できます ．

TskFuzzyRuleType rule1 = new TskFuzzyRuleType("rule1", "or","MAX", 1.0f);

AntecedentType ant1 = new AntecedentType();

ant1.addClause(new ClauseType(food, rancid));

ant1.addClause(new ClauseType(service, poor));

rule1.setAntecedent(ant1);

TskConsequentType con1 = new TskConsequentType();

con1.addTskThenClause(tip, cheap);

rule1.setTskConsequent(con1);

rbTsk.addTskRule(rule1);

### AnYaRuleType

AnYaRuleTypeオブジェクトは、IF-THEN-ELSEの形式でAnYaルールを表します。各ルールには3つの属性があります。

* Name : ルールの一意の名前を持つstringです．この情報はルールを生成するとき必要です．
* *weight：* 推論エンジンで使用されるルールの重要性を表すfloatです。この情報はオプションであり、その領域は[0、1]で、デフォルト値は1.0です。
* *NetworkAddress*：コンピュータネットワーク内のルールの場所を示すstringです。ルールを作成するとき、この情報は任意です。デフォルト値は "127.0.0.1"です。

例えば，簡単なルールの定義は以下のようになります．

AnYaRuleType rule1 = new AnYaRuleType("rule1");

各AnYaRuleTypeオブジェクトは、1つのAnYaAntecedentTypeオブジェクトと1つのConsequentTypeオブジェクトまたはTskConsequentTypeオブジェクトで構成されています。 AnYaAntecedentTypeオブジェクトは、datacloudに基づくルールの先行部分を表し、それぞれには、ルールで使用されるdatacloudを表す1つのAnYaDataCloudTypeオブジェクトを持ちます．

したがって、ルールの先行部分と後続部分を次のように定義できます

AnYaRuleType rule1 = new AnYaRuleType("rule1");

AnYaAntecedentType ant1 = new AnYaAntecedentType();

ant1.setDataCloudName(foodRancidServicePoor);

rule1.setAnYaAntecedent(ant1);

ConsequentType con1 = new ConsequentType();

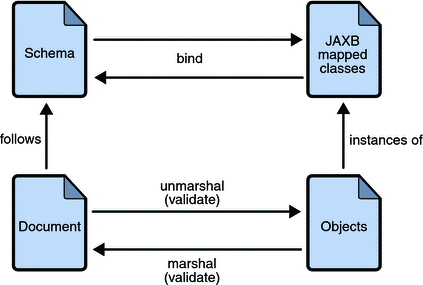
con1.addThenClause(tip, cheap);

rule1.setConsequent(con1);

rbAnY.addRule(rule1);

### Load/Write FML Files

IEEE規格に従ってファジーシステムを読み込み、書き込むことができます。 JFMLは、新しい[Java API JAXB（XML 構築用Javaアーキテクチャ）](https://docs.oracle.com/javase/tutorial/jaxb/intro/index.html)を利用します。これは、XML schemaとJava representationを高速で便利に結びつけることを可能にし、Java開発者がXMLデータと処理機能をJavaアプリケーションに簡単に組み込むことができるようにします。



次のJavaコードを使用して、JFMLでIEEE標準に従ってXMLファイルを読み取る(復元)ことができます，

File xml = new File("./XMLFiles/tipper.xml");

FuzzyInferenceSystem tipper = JFML.load(xml);

JFMLを使用して作成したファジーシステムは、次のJavaコードを使用して、IEEE標準に従ってXMLファイルに記述（変換）できます

File xml = new File("./XMLFiles/ourFS.xml");

JFML.writeFSTtoXML(fs, xml);

FMLコードは、FMLプログラムのXSLTベースの変換とJavaベースのFMLコードの実行という2つの異なるアプローチを考慮して実行できることに注意してください。最初の場合では、XSLTエンジンがFMLコードを、システムのFML記述の単純なテキスト操作によってファジーシステムを記述するJavaプログラムに変換します。 2番目の場合では、Javaメソッドは、FMLコードを読み取り、関連するファジーシステムを記述するJavaオブジェクトの階層を自動的に構築するように設計されています。ファジー推論システムを開発するためのこれら2つのアプローチ間の開発努力の観点から、2番目のアプローチは、最初のアプローチよりも使いやすいように見えます。実際、最初の場合では、プログラマーはまずXSLTエンジンを適用して関連するJavaコードを生成し、次にを適用する必要があります

Notice that FML code can be executed by considering two different approaches: a XSLT-based translation of FML programs, and a Java-based execution of FML code. In the first case, an XSLT engine translates an FML code into a Java program describing fuzzy systems by means of a pure textual manipulation of the FML description of the systems. In the second case, a Java method has been opportunely designed to read the FML code and automatically build a hierarchy of Java objects describing the related fuzzy systems. In terms of development effort between these two approaches for developing fuzzy inference systems, the second approach appears to be more convenient to use than the first one. Indeed, in the first case, a programmer needs first to apply the XSLT engine and thus generating the related Java code, then

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**Java Fuzzy Markup Language. ©2019** 29

compiling the Java code to produce a Java bytecode ready to interpret and to run. In the second case, it is sufficient for a programmer to load the FML code in his/her Java application, thus automatically obtaining a collection of Java objects ready to use with the Java-based fuzzy inference engine in order to effectively run the designed fuzzy system.

JFML also allows us to import/export a fuzzy system to different formats. JFML includes a module to read FLSs from FCL or PMML documents, and to write a FLS designed with JFML in a FCL or PMML document. Moreover, this library also includes a module to read FLSs designed with the Matlab Fuzzy Logic Toolbox and to export FLSs designed with JFML to the Matlab toolbox. In the case of reading a FCL document, according to **Fuzzy Control Language (FCL) specification IEC 61131 part 7**, we must use the ImportFCL class in JFML, which allows us to unmarshal the data of the file in a FuzzyInferenceSystem object. Next Java code shows an example of reading a FCL fuzzy system:

ImportFCL fcl = new ImportFCL(); FuzzyInferenceSystem fs = fcl.importFuzzySystem("./XMLFiles/robot.fcl");

In the case of reading a FLS designed with Matlab (.fis file), we must use the ImportMatlab class in JFML:

ImportMatlab fis = new ImportMatlab(); FuzzyInferenceSystem fs = fis.importFuzzySystem("./XMLFiles/IrisMamdani1.fis");

In the case of reading a FLS in PMML format, we must use the ImportPMML class in JFML:

ImportPMML pmml = new ImportPMML(); FuzzyInferenceSystem fs = pmml.importFuzzySystem("./XMLFiles/TipperTSK.frbsPMML");

Similarly, we can export a FLS designed with JFML to other formats. For example, next Java code shows an example of writing a FLS object (fs) to FCL:

ExportFCL fcl = new ExportFCL(); fcl.exportFuzzySystem(fs, "./XMLFiles/robot.fcl");

to Matlab format (.fis file):

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**Java Fuzzy Markup Language. ©2019** 30

ExportMatlab fis = new ExportMatlab(); fis.exportFuzzySystem(fs, "./XMLFiles/IrisMamdani1.fis");

and to PMML:

ExportPMML pmml = new ExportPMML(); pmml.exportFuzzySystem(fs, "./XMLFiles/TipperTSK.frbsPMML");

**Fuzzy Inference**

We can easily evaluate a fuzzy system in JFML. We must set values for the input variables, evaluate the system (fuzzy inference), and get the output values. Next Java code is a simple example in which we evaluate a fuzzy system for the tipper problem.

public class EvaluateTipperExample {

public static void main(String[] args) {

// Loading Fuzzy System from an XML file according the standard IEEE 1855

File xml = new File("./XMLFiles/tipper.xml"); FuzzyInferenceSystem tipper = JFML.load(xml); // Set inputs values KnowledgeBaseVariable food = fs.getVariable("food");

KnowledgeBaseVariable service = fs.getVariable("service");

food.setValue(6); service.setValue(8); // Fuzzy inference tipper.evaluate(); // Get output KnowledgeBaseVariable tip = fs.getVariable("tip"); float value = tip.getValue(); // Printing results System.out.println("RESULTS"); System.out.println(" (INPUT): " + food.getName() + "=" + food.getValue() + ", " + service.getName() + "=" + service.getValue());

System.out.println(" (OUTPUT): " + tip.getName() + "=" + value);

// Printing the FuzzySystem

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**Java Fuzzy Markup Language. ©2019** 31

System.out.println(fs.toString()); } }

Moreover, this Java code also prints the definition of the fuzzy system (Knowledge base + Rule bases) making use of the method toString of the FuzzyInferenceSystem class. Notice that the method toString is available in most of the classes of JFML.

**Extensibility**

The modular design of JFML allows us to extend the API easily. Moreover, we can add new elements making use of the custom elements that have the classes, according to the pattern [custom\_\S\*]. This extension mechanism allows to define new elements without changing the language grammar.

For instance, if we want to introduce a new T-norm in JFML (ie, the T-norm TN), we must include the following changes in the AndLogicalType class:

Include in the method "operate" the option for the new T-norm:

public float operate(float x, float y) {

String op = getOperator(); if (op.equals(StandardAndMethodType.MIN.value()))

return min(x, y); else if (op.equals(StandardAndMethodType.PROD.value()))

return prod(x, y); else if (op.equals(StandardAndMethodType.BDIF.value()))

return bdif(x, y); else if (op.equals(StandardAndMethodType.DRP.value()))

return drp(x, y); else if (op.equals(StandardAndMethodType.EPROD.value()))

return eprod(x, y); else if (op.equals(StandardAndMethodType.HPROD.value()))

return hprod(x, y); else if (op.equals(StandardAndMethodType.NILMIN.value()))

**Java Fuzzy Markup Language. ©2019** 32

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

return nilmin(x, y); **else if (op.equals("custom\_TN"))**

**return custom\_TN(x, y);** else

return min(x, y); }Add the method custom\_TN to the class, which implements the T-norm TN:

private float custom\_TN(float x, float y) {

// TODO }

Now, we could use the T-norm custom\_TN in the definitions of our fuzzy systems according to the IEEE standard. On the other hand, The modular design allows us to easily extend JFML with new functions. For instance, JMFL includes the class ImportFCL that allows us to import FCL fuzzy systems.

With the aim of having a code free of bugs we recommend running unit tests when coding new functions for JFML. In short, **unit testing** consists of writing short code fragments which are aimed at testing that the new function works properly.

Once JFML is extended, it is mandatory to check not only that the new funcitons work properly but also that everything is still in accordance with the IEEE Std 1855. With that aim, we recommend running the provided **examples**.

**Embedded Systems**

The embedded system module assists developers in the design and implementation of fuzzy systems for open hardware embedded systems. The JFML is now ready for Arduino and Raspberry Pi with this module, but it can be easily extended to other hardware architectures. Moreover, the module supports several connection types (WiFi, Bluetooth and USB) in order to make feasible running fuzzy systems in a remote computer when, due to hardware limitations, it is not possible to run the fuzzy systems locally in the embedded systems. In addition, the module allows to automatically generate runnable files on Arduino or Raspberry Pi in order to support non-expert users, ie, users without specific knowledge about embedded systems or without strong programming skills.

The main class is EmbeddedSystem wich is responsible for defining both the characteristics and the type of connection with the embedded system. It requires a

**Java Fuzzy Markup Language. ©2019** 33

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

name, a type of connection (WiFi, Bluetooth and USB) and a baud rate for establishing the connection with the JFML core. It has also associated a list of EmbeddedVariable objects which are in charge of the association between variables of the knowledge base and sensors/actuators. The classes EmbeddedSystemArduino and EmbeddedSystemRaspberry, which extend the abstract class EmbeddedSystem, are used for Arduino-based and RaspberryPi-based implementations. Arduino boards can be connected to serial ports (USB or Bluetooth) or WiFi and used for designing FLCs in a simple way. These different connections are taken into account by the EmbeddedSystemArduinoUSB, EmbeddedSystemArduinoBluetooth or EmbeddedSystemArduinoWIFI classes which extend the EmbeddedSystemArduino class. Similarly, the classes EmbeddedSystemRaspberryUSB, EmbeddedSystemRaspberryBluetooth or EmbeddedSystemRasperryWIFI, which extend the EmbeddedSystemRaspberry class, are in charge of defining the different connections between the Raspberry Pi and the JFML.

For instance, if we want to create an Arduino-based embedded system connected via USB with 2 sensors, we can do it as follow:

Associating variables from the knowledge base to the sensors. For example, a ultrasonic HCSR04 sensor and a LED:

Sensor sensor1 = new ArduinoHC\_SR04("name\_sensor", ArduinoPin.PIN\_1, ArduinoPin.PIN\_2); Sensor sensor2 = new ArduinoLED("name\_sensor", ArduinoPin.PIN\_3);

ArrayList embeddedVariables = new ArrayList<>(); embeddedVariables.add(new EmbeddedVariableArduino(0,variable1,sensor1)); embeddedVariables.add(new EmbeddedVariableArduino(1,variable2,sensor2));

Creating an embedded system instance of Arduino with the sensors defined bellow and connected via USB to 9600 baud rate:

EmbeddedSystem arduinoUSB; arduinoUSB = new EmbeddedSystemArduinoUSB ("name\_embedded","USB\_PORT", 9600, embeddedVariables);

Or, for example, if we want to create a RaspberryPi-based embedded system connected via WiFi with a DHT22 temperature sensor, we can do it as follow:

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**Java Fuzzy Markup Language. ©2019** 34

Associating variables from the knowledge base to the sensors. For example, a ultrasonic HCSR04 sensor and a LED:

Sensor sensor1 = new RaspberryDHT22("name\_sensor", RaspberryPin.PIN\_1, RaspberryPin.PIN\_2);

ArrayList embeddedVariables = new ArrayList<>(); embeddedVariables.add(new EmbeddedVariableArduino(0,variable1,sensor1));

Creating an embedded system instance of RaspberryPi with the DHT22 sensor defined bellow and connected via WiFi:

EmbeddedSystem rpiWIFI; rpiWIFI = new EmbeddedSystemRasperryWIFI ("name\_embedded","IP\_ADDRESS", "SSID", "PASSWORD", embeddedVariables);

Now, we can run an instanceof a FLS on the arduino board and/or the RaspberryPi iteratively or during a single iteration. The EmbeddedController class is in charge of coordinating all the embedded systems connected to the JFML. As many embedded systems as the designer desires could be included in a unique EmbeddedController instance. For example, if we want to run a FLS indefinitely on the two embedded systems created before, we have to do the following:

ArrayList boards = new ArrayList<>(); boards.add(arduinoUSB); boards.add(rpiWIFI);

EmbeddedController controller = new EmbeddedControllerSystem(boards, FLS); controller.run();

Once a embedded system instance is created in JFML, a runnable file according to the architecture design requirements of the embedded system is necessary. The EmbeddedSystem class defines an abstract method to create automatically this file where a user without specific knowledge about the embedded system architecture or programming language could easily run a FLC on it. For example, to create a runnable file for the previously defined Arduino board, we can do it as follow:

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**Java Fuzzy Markup Language. ©2019** 35

arduinoUSB.createRunnableEmbeddedFile("name\_file");

**Sensors**

A collection of sensors of the most commonly used Arduino-based and Raspberry Pi- based solutions is already available within the embedded system JFML module. Users can add more sensors and/or actuators by accessing the programming code and extending the Sensor class. The module includes classic sensors such as the temperature and humidity sensor DHT22, the ultrasonic sensor HC-SR04, the motion sensor H-SR501, the Gas sensor MQ-2, the light sensor LDR, and the accelerometer/gyroscope MPU6050; and actuators such as LEDs, DC motor and driver motor. All these sensors are implemented and users can easily used them without any additional programming task.

Temperature DHT22 sensor: default constructor

Sensor sensorArduino = new ArduinoDHT22\_temperature("name", ArduinoPin.PIN\_XX); Sensor sensorRaspberry = new RaspberryDHT22\_temperature("name", RaspberryPin.GPIOXX);

Humidity DHT22 sensor: default constructor

Sensor sensorArduino = new ArduinoDHT22\_humidity("name", ArduinoPin.PIN\_XX); Sensor sensorRaspberry = new RaspberryDHT22\_humidity("name", RaspberryPin.GPIOXX);

Ultrasonic HC\_SR04 sensor: default constructor

Sensor sensorArduino = new ArduinoHC\_SR04("name", ArduinoPin.PIN\_XX, ArduinoPin.PIN\_XX); Sensor sensorRaspberry = new RaspberryHC\_SR04("name", RaspberryPin.GPIOXX, RaspberryPin.GPIOXX);

Motion HC\_SR501 sensor: default constructor

Sensor sensorArduino = new ArduinoHC\_SR501("name", ArduinoPin.PIN\_XX, timeout); Sensor sensorRaspberry = new RaspberryHC\_SR501("name", RaspberryPin.GPIOXX, timeout);

Light LDR sensor: default constructor

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**Java Fuzzy Markup Language. ©2019** 36

Sensor sensorArduino = new ArduinoLIGHT("name", ArduinoPin.PIN\_XX); Sensor sensorRaspberry = new RaspberryLIGHT("name", RaspberryPin.GPIOXX);

LED PWM sensor: default constructor

Sensor sensorArduino = new ArduinoLED\_PWM("name", ArduinoPin.PIN\_XX); Sensor sensorRaspberry = new RaspberryLED\_PWM("name", RaspberryPin.GPIOXX);

Controller motor L298N actuator: default constructor

Sensor actuatorArduino = new ArduinoH\_BRIDGE\_L298N(String name, ArduinoPin ENA, ArduinoPin IN1, ArduinoPin IN2, ArduinoPin IN3, ArduinoPin IN4, ArduinoPin ENB, int min, int max, int minSpeed, int maxSpeed); Sensor actuatorRaspberry = new ArduinoH\_BRIDGE\_L298N(String name, ArduinoPin ENA, ArduinoPin IN1, ArduinoPin IN2, ArduinoPin IN3, ArduinoPin IN4, ArduinoPin ENB, int min, int max, int minSpeed, int maxSpeed);

DC motor actuator: default constructor

Sensor actuatorArduino = new ArduinoDCMOTOR\_PWM("name", ArduinoPin.PIN\_XX); Sensor actuatorRaspberry = new RaspberryDCMOTOR\_PWM("name", RaspberryPin.GPIOXX);

Servo SG90 actuator: default constructor

Sensor actuatorArduino = new ArduinoSERVO("name", ArduinoPin.PIN\_XX); Sensor actuatorRaspberry = new RaspberrySERVO("name", RaspberryPin.GPIOXX);

Several real word problems can be found in the **Examples**. section.

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**Java Fuzzy Markup Language. ©2019** 37

**Command line**

The JFML library can be freely downloaded from **GitHub**. A folder with a runnable version of the JFML Library (compiled with JRE v8) and several XML files as examples can be found in the Examples folder.

The JFML library can be run with 3 main arguments (ProblemName InferenceExample DataFile) but brackets are not required (in JFML-vx.x.jar, x corresponds with the version of the library):

Usage: java -jar JFML-vx.x.jar [options]

Options: Tipper [Mamdani1 | Mamdani2 | Mamdani3 | TSK | Tsukamoto1 | Tsukamoto2 | AnYa] test-data-file

Options: JapaneseDietAssessment Mamdani test-data-file Options: Iris [Mamdani1 | Mamdani2 | Mamdani3] test- data-file

Options: InvertedPendulum [Mamdani1 | Mamdani2 | TSK1 | TSK2] test-data-file

Options: Robot Mamdani test-data-file

You can also run the library with a specific instance as follows:

Options: ProblemName InferenceExample V1 D1 V2 D2 ... ProblemName: Tipper, JapaneseDietAssessment, etc. InferenceExample; Mamdani, Mamdani1, Mamdani2, TSK, AnYa, etc.

Notice that the combination of ProblemName and InferenceExample must be in accordance with the name of an XML file in the folder ./XMLFiles. You must be also sure of providing the entire list of pairs variable name (Vi, as it is in the XML file) and numerical value (Di) for evaluation.

Example:

java -jar ./lib/JFML-vx.x.jar Iris Mamdani2 SepalLength 5.1 SepalWidth 3.5 PetalLength 1.4 PetalWidth 0.2

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**Java Fuzzy Markup Language. ©2019** 38

You can also run the library with a specific instance, for your own XML file, considering the options: XMLfilePath NbOutputs ON1 ON2 ... V1 D1 V2 D2 ...

Example:

java -jar ./lib/JFML-vx.x.jar ./XMLFiles/RobotIEEEstd1855.xml 2 la av rd 0.2 dq 0.25 o 20 v 0.25

**Running the examples**

Before running the examples, please be sure the XMLFiles folder contains the required XML files. Otherwise, you can create some of them from the command line: Examples:

java -classpath ./lib/JFML-vx.x.jar jfml.test.CreateIrisMamdaniExampleXML1

java -classpath ./lib/JFML-vx.x.jar jfml.test.CreateIrisMamdaniExampleXML2

java -classpath ./lib/JFML-vx.x.jar jfml.test.CreateIrisMamdaniExampleXML3

java -classpath ./lib/JFML-vx.x.jar jfml.test.CreateInvertedPendulumMamdaniExampleXML1

java -classpath ./lib/JFML-vx.x.jar jfml.test.CreateInvertedPendulumMamdaniExampleXML2

java -classpath ./lib/JFML-vx.x.jar jfml.test.CreateInvertedPendulumTSKExampleXML1

java -classpath ./lib/JFML-vx.x.jar jfml.test.CreateInvertedPendulumTSKExampleXML2

java -classpath ./lib/JFML-vx.x.jar jfml.test.CreateJapaneseDietAssessmentMamdaniExampleXML

java -classpath ./lib/JFML-vx.x.jar jfml.test.CreateTipperMamdaniExampleXML1

java -classpath ./lib/JFML-vx.x.jar jfml.test.CreateTipperMamdaniExampleXML2

java -classpath ./lib/JFML-vx.x.jar jfml.test.CreateTipperMamdaniExampleXML3

java -classpath ./lib/JFML-vx.x.jar jfml .test.CreateTipperTSKExampleXML

java -classpath ./lib/JFML-vx.x.jar

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**Java Fuzzy Markup Language. ©2019** 39

jfml.test.CreateTipperTsukamotoExampleXML1

java -classpath ./lib/JFML-vx.x.jar jfml.test.CreateTipperTsukamotoExampleXML2

java -classpath ./lib/JFML-vx.x.jar jfml.test.CreateTipperAnYaExampleXML

Test:

java -jar ./lib/JFML-vx.x.jar Iris Mamdani1 ./XMLFiles/test-data-Iris1.txt

java -jar ./lib/JFML-vx.x.jar Iris Mamdani1 PetalWidth 0.2 java -jar ./lib/JFML-vx.x.jar Iris Mamdani2 ./XMLFiles/test-data-Iris2.txt

java -jar ./lib/JFML-vx.x.jar Iris Mamdani2 SepalLength 5.1 SepalWidth 3.5 PetalLength 1.4 PetalWidth 0.2

java -jar ./lib/JFML-vx.x.jar Iris Mamdani3 ./XMLFiles/test-data-Iris1.txt

java -jar ./lib/JFML-vx.x.jar Iris Mamdani3 PetalWidth 0.5 java -jar ./lib/JFML-vx.x.jar InvertedPendulum Mamdani1 ./XMLFiles/test-data-InvertedPendulum.txt

java -jar ./lib/JFML-vx.x.jar InvertedPendulum Mamdani2 ./XMLFiles/test-data-InvertedPendulum.txt

java -jar ./lib/JFML-vx.x.jar InvertedPendulum TSK1 ./XMLFiles/test-data-InvertedPendulum.txt

java -jar ./lib/JFML-vx.x.jar InvertedPendulum TSK 2 ./XMLFiles/test-data-InvertedPendulum.txt

java -jar ./lib/JFML-vx.x.jar JapaneseDietAssessment Mamdani ./XMLFiles/test-data-JapaneseDietAssessment.txt

java -jar ./lib/JFML-vx.x.jar Tipper Mamdani1 ./XMLFiles/test-data-Tipper1.txt

java -jar ./lib/JFML-vx.x.jar Tipper Mamdani2 ./XMLFiles/test-data-Tipper1.txt

java -jar ./lib/JFML-vx.x.jar Tipper Mamdani3 ./XMLFiles/test-data-Tipper2.txt

java -jar ./lib/JFML-vx.x.jar Tipper TSK ./XMLFiles/test- data-Tipper1.txt

java -jar ./lib/JFML-vx.x.jar Tipper Tsukamoto1 ./XMLFiles/test-data-Tipper1.txt

java -jar ./lib/JFML-vx.x.jar Tipper Tsukamoto2

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**Java Fuzzy Markup Language. ©2019** 40

./XMLFiles/test-data-Tipper1.txt

java -jar ./lib/JFML-vx.x.jar Tipper AnYa ./XMLFiles/test- data-Tipper2.txt

java -jar ./lib/JFML-vx.x.jar Robot Mamdani ./XMLFiles/test-data-Robot.txt

java -jar ./lib/JFML-vx.x.jar ./XMLFiles/RobotIEEEstd1855.xml ./XMLFiles/test-data-Robot.txt

java -jar ./lib/JFML-vx.x.jar ./XMLFiles/RobotIEEEstd1855.xml 2 la av rd 0.2 dq 0.25 o 20 v 0.25

Export:

java -classpath ./lib/JFML-vx.x.jar jfml.test.ExportExample ./XMLFiles/IrisMamdani1.xml ./XMLFiles/IrisMamdani1.fis

java -classpath ./lib/JFML-vx.x.jar jfml.test.ExportExample ./XMLFiles/IrisMamdani1.xml ./XMLFiles/IrisMamdani1.fcl

java -classpath ./lib/JFML-vx.x.jar jfml.test.ExportExample ./XMLFiles/IrisMamdani2.xml ./XMLFiles/IrisMamdani2.fis

java -classpath ./lib/JFML-vx.x.jar jfml.test.ExportExample ./XMLFiles/TipperMamdani1.xml ./XMLFiles/TipperMamdani1.fis

java -classpath ./lib/JFML-vx.x.jar jfml.test.ExportExample ./XMLFiles/TipperMamdani1.xml ./XMLFiles/TipperMamdani1.fcl

java -classpath ./li b/JFML-vx.x.jar jfml.test.ExportExample ./XMLFiles/TipperMamdani2.xml ./XMLFiles/TipperMamdani2.fis

java -classpath ./lib/JFML-vx.x.jar jfml.test.ExportExample ./XMLFiles/TipperMamdani2.xml ./XMLFiles/TipperMamdani2.fcl

java -classpath ./lib/JFML-vx.x.jar jfml.test.ExportExample ./XMLFiles/TipperTSK.xml ./XMLFiles/TipperTSK.fis

java -classpath ./lib/JFML-vx.x.jar jfml.test.ExportExample ./XMLFiles/TipperTSK.xml ./XMLFiles/TipperTSK.fcl

java -classpath ./lib/JFML-vx.x.jar jfml.test.ExportExample ./XMLFiles/IrisMamdani2.xml ./XMLFiles/IrisMamdani2.frbsPMML

java -classpath ./lib/JFML-vx.x.jar jfml.test.ExportExample ./XMLFiles/TipperTSK.xml ./XMLFiles/TipperTSK.frbsPMML

Import:

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**Java Fuzzy Markup Language. ©2019** 41

java -classpath ./lib/JFML-vx.x.jar jfml.test.ImportExample ./XMLFiles/IrisMamdani1.fis ./XMLFiles/IrisMamdani1.xml

java -classpath ./lib/JFML-vx.x.jar jfml.test.ImportExample ./XMLFiles/IrisMamdani1.fcl ./XMLFiles/IrisMamdani1.xml

java -classpath ./lib/JFML-vx.x.jar jfml.test.ImportExample ./XMLFiles/IrisMamdani2.fis ./XMLFiles/IrisMamdani2.xml

java -classpath ./lib/JFML-vx.x.jar jfml.test.ImportExample ./XMLFiles/TipperMamdani1.fis ./XMLFiles/TipperMamdani1.xml

java -classpath ./lib/JFML-vx.x.jar jfml.test.ImportExample ./XMLFiles/TipperMamdani1.fcl ./XMLFiles/TipperMamdani1.xml

java -classpath ./lib/JFML-vx.x.jar jfml.test.ImportExample ./XMLFiles/TipperMamdani2.fis ./XMLFiles/TipperMamdani2.xml

java -classpath ./lib/JFML-vx.x.jar jfml.test.ImportExample ./XMLFiles/TipperMamdani2.fcl ./XMLFiles/TipperMamdani2.xml

java -classpath ./lib/JFML-vx.x.jar jfml.test.ImportExample ./XMLFiles/TipperTSK.fis ./XMLFiles/TipperTSK.xml

java -classpath ./lib/JFML-vx.x.jar jfml.test.Import Example ./XMLFiles/TipperTSK.fcl ./XMLFiles/TipperTSK.xml

java -classpath ./lib/JFML-vx.x.jar jfml.test.ImportExample ./XMLFiles/IrisMamdani2.frbsPMML ./XMLFiles/IrisMamdani2.frbsPMML.xml

java -classpath ./lib/JFML-vx.x.jar jfml.test.ImportExample ./XMLFiles/TipperTSK.frbsPMML ./XMLFiles/TipperTSK.frbsPMML.xml

Circular Definitions:

java -classpath ./lib/JFML-vx.x.jar jfml.test.TaoExample

In addition, notice that you can run the examples above by calling to the related bat files in Windows OS or by running the Makefile otherwise. For example, in Windows OS, running the file main-jfml- CreateIrisMamdaniExampleXML1.bat you can create the file TipperMamdani1.xml. The same example can be executed in Mac or Linux by running the makefile: make createTipperMamdani1. In the same way, you can test the examples by running the bat file main-jfml-test.bat and the corresponding parameters.

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**Java Fuzzy Markup Language. ©2019** 42

**Py4JFML**

**Py4JFML** is freely accessible in **GitHub** from **https://github.com/cmencar/py4jfml**

1. Download the file py4jfml-xytar.gz in the dist folder (xy stands for the current version, eg 1.0) 2. run $ pip3 install py4jfml-xytar.gz The installation program will automatically download and install py4j if not available in the system. If you want to test the library, you can extract the testlib folder from the archive and execute one of the contained scripts. For example: $ python3 CreateInvertedPendulumMamdaniExampleXML1.py In Windows, a Py4JNetworkError could arise when executing **Py4JFML**. This is a known issue of py4j and occurs when the library tries to shut down the py4j server before terminating. This problem does not affect the functionality of the library, but you may get a memory leakage due to the server still running after the program termination.

**Examples**

JFML includes several examples for both users and developers in the package **jfml.test**. On the one hand, the related Java files are provided in the folder src/jfml/test. On the other hand, the related XML files are provided in the folder XMLFiles. Both folders can be freely downloaded from **GitHub**.

**Iris (Classification Problem) TAO (Classification Problem) Inverted Pendulum (Control Problem) Tipper (Regression Problem) Japanese Diet Assessment (Regression Problem) Robot (Real-world Problem)**

Let us briefly introduce below these problems as well as the related FLS that we have developed just for illustrative purpose:

**Iris (Classification Problem)**

This is likely to be the best known classification problem in the pattern recognition community. The original dataset was reported by RA Fisher in 1936. The whole dataset is available at **UCI**. It includes 4 input variables (SepalLength, SepalWith, PetalLength, and PetalWidth) and 1 output variable (Iris Class).

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**Java Fuzzy Markup Language. ©2019** 43

We have developed 3 Mamdani-type FLS:

**Mamdani1.**

This FLS comprises only 1 input variable (PetalWidth) and 1 output variable (Iris Class). The input is described by a strong fuzzy partition with 3 linguistic terms (low, medium, high) which are implemented as Trapezoidal (**FuzzyTermType.TYPE\_trapezoidShape**) membership functions. In addition, the rule base is implemented as **MamdaniRuleBaseType** and it is made up of 3 rules, one per output class. It applies the usual Mamdani min-max inference mechanism and the well-known **Mean of Max** as defuzzification mechanism.

Related files:

CreateIrisMamdaniExampleXML1.java IrisMamdani1.xml test-data-Iris1.txt

**Mamdani2.**

This FLS considers all variables related to the original iris dataset, ie, 4 inputs and 1 output. The inputs are described by strong fuzzy partitions with 3 or 4 linguistic terms (the fourth one is defined as the negation of one of the others) which are implemented as Triangular (**FuzzyTermType.TYPE\_triangularShape**) or Trapezoidal (**FuzzyTermType.TYPE\_trapezoidShape**) membership functions. In addition, the rule base is implemented as **MamdaniRuleBaseType** and it is made up of 3 rules, one per output class. Like in the previous case, we apply the usual Mamdani min-max inference mechanism and the well-known **Mean of Max** as defuzzification mechanism.

Related files:

CreateIrisMamdaniExampleXML2.java IrisMamdani2.xml test-data-Iris2.txt

**Mamdani3.**

This FLS is dual to Mamdani1. The main difference arises from the fact that the input variable contains now 12 linguistic terms which are implemented with some of the types of membership function recognized by the IEEE Std 1855-2016:

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**Java Fuzzy Markup Language. ©2019** 44

lowLIN (**leftLinearShape**) lowGAU (**leftGaussianShape**) lowPi (**piShape**) lowZ (**zShape**) mediumTRI (**triangularShape**) mediumTRA (**trapezoidShape**) mediumGAU (**gaussianShape**) mediumREC (**rectangularShape**) highLIN (**rightLinearShape**) highGAU (**rightGaussianShape**) highSIN (**singletonShape**) highS (**sShape**)

In addition, the rule base includes 12 rules, one per linguistic term enumerated above. Namely, each rule in Mamdani1 is substituted here by 4 equivalent rules where the only difference is the membership function appearing in the antecedent of the rule. Moreover, we apply the usual Mamdani min-max inference mechanism and the well-known **Center of Gravity** as defuzzification mechanism.

Related files:

CreateIrisMamdaniExampleXML3.java IrisMamdani3.xml test-data-Iris1.txt

**TAO (Classification Problem)**

This is a synthetic problem with two input variables (x1 and x2) and two classes (C0 and C1). The TAO dataset comes from sampling the TAO figure with 1888 instances equally spaced along the horizontal and vertical axis. Each instance has 2 real valued attributes (x1 and x2) which correspond to the [x,y] coordinates, and the associated class which is related to the color of the figure in the target point. The interested reader is kindly referred to the following papers for further details:

D. Garcia, A. Gonzalez, and R. Perez, "A two-step approach of feature construction for a genetic learning algorithm," in IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), Taipei, Taiwan, 2011, pp. 1255–1262. E. Bernado, X. Llora, and JM Garrell, "XCS and GALE: A Comparative Study of Two Learning Classifier Systems," in Data Mining. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 115–132

We have developed 1 Mamdani-type FLS:

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**Java Fuzzy Markup Language. ©2019** 45

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**Mamdani TAO.**

This FLS initially comprises 2 inputs and 1 classification output. In addition, a third input variable was created from the addition of the two input variables (Sum(x1,x2)). We designed homogeneous fuzzy partitions with 5 triangular fuzzy sets (Very Low, Low, Medium, High Very High) for the three input variables. With respect to the fuzzy reasoning mechanism, the minimum t-norm plays the role of both the AND operator and the accumulation method, and maximum t-conorm plays the role of both the OR operator and the activation method. The rule base consists of 3 Disjunctive Normal Form (DNF) rules with weights. The classifier is first initialized with the Matlab Fuzzy Logic Toolbox, then exported to FML. Finally, we used JFML to create R2 and R3 which required circular definitions. The final system is exported again to Matlab.

Related files:

TaoExample.java taoMatlab.fis taoFML.xml taoFML.fis

**Inverted Pendulum (Control Problem)**

This is a well-known problem in the control field. It consists of moving horizontally a cart and a pole as an inverted pendulum, ie, a pendulum with the centre of mass above its pivot point, while it keeps a vertical position. We address the case of having 2 input variables (Angle and ChangeAngle) and 1 output variable (Force).

We have developed 2 Mamdani-type and 2 TSK-type FLS:

**Mamdani1.**

This FLS comprises 2 inputs and 1 regression output. Each variable (no matter input or output) is characterized by a strong fuzzy partition with 5 basic linguistic terms (very negative, negative, zero, positive, very positive) which are implemented as Triangular (**FuzzyTermType.TYPE\_triangularShape**) or Trapezoidal (**FuzzyTermType.TYPE\_trapezoidShape**) membership functions. In addition, in the case of the inputs, we defined 2 additional OR composite terms (very negative or negative, positive or very positive) which are implemented as Trapezoidal (**FuzzyTermType.TYPE\_trapezoidShape**) membership functions. Accordingly, the rule base is implemented as

**Java Fuzzy Markup Language. ©2019** 46

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**MamdaniRuleBaseType** and it is made up of 19 rules. It applies the usual Mamdani min-max inference mechanism and the **Center of Gravity** as defuzzification mechanism.

Related files:

CreateInvertedPendulumMamdaniExampleXML1.java InvertedPendulumMamdani1.xml test-data-InvertedPendulum.txt

**Mamdani2.**

This FLS is dual to Mamdani1. The only difference arises from the fact that now the OR composite terms are implemented with the classes **OrLogicalType** and **CircularDefinitionType**.

Related files:

CreateInvertedPendulumMamdaniExampleXML2.java InvertedPendulumMamdani2.xml test-data-InvertedPendulum.txt

**TSK1.**

This FLS is dual to Mamdani2 but now the output is defined as a **TskVariableType** and the 5 associated linguistic terms are implemented as order-0 **TskTermType**.

Related files:

CreateInvertedPendulumTSKExampleXML1.java InvertedPendulumTSK1.xml test-data-InvertedPendulum.txt

**TSK2.**

This FLS is dual to TSK1. The only difference arises from the fact that now the output variable includes linguistic terms defined by both order-0 and order-1 **TskTermType**

Related files:

CreateInvertedPendulumTSKExampleXML2.java InvertedPendulumTSK2.xml test-data-InvertedPendulum.txt

**Java Fuzzy Markup Language. ©2019** 47

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**Tipper (Regression Problem)**

This problem consists of computing the right tip in a restaurant. It considers two inputs (food and service) and one output (tip).

We have developed 3 Mamdani-type, 1 TSK-type, 2 Tsukamoto-type and 1 AnYa-type FLS:

**Mamdani1.**

This FLS comprises 2 inputs and 1 regression output. Each variable (no matter input or output) is characterized by a strong fuzzy partition.

- food is characterized by 2 linguistic terms (rancid, delicious): rancid which is implemented as a Triangular (**FuzzyTermType.TYPE\_triangularShape**) membership function and delicious which is implemented as a rightLinear (**FuzzyTermType.TYPE\_rightLinearShape**) membership function.

- service is characterized by 3 linguistic terms (poor, good, excellent): poor which is implemented as a leftGaussian (**FuzzyTermType.TYPE\_leftGaussianShape**) membership function, good which is implemented as a Gaussian (**FuzzyTermType.TYPE\_gaussianShape**) membership function, and excellent which is implemented as a rightGaussian (**FuzzyTermType.TYPE\_rightGaussianShape**) membership function.

- tip is characterized by 3 linguistic terms (cheap, average, generous) which are implemented as Triangular (**FuzzyTermType.TYPE\_triangularShape**) membership functions. In addition, the rule base is implemented as **MamdaniRuleBaseType** and it is made up of 3 rules. We also apply the well-known **Center of Gravity** as defuzzification mechanism. However, contrary to the usual Mamdani rules, here rule premises are connected through or (MAX) instead of and (MIN). Moreover, different from previous examples, here in rule1 we can see now to use the edge very.

Related files:

CreateTipperMamdaniExampleXML1.java TipperMamdani1.xml test-data-Tipper1.txt

**Java Fuzzy Markup Language. ©2019** 48

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**Mamdani2.**

This FLS is dual to Mamdani1. The only difference arises from the fact that now the linguistic term cheap in the output variable tip is implemented with the class **PointSetShapeType**. Moreover, we use Lagrange as **InterpolationMethodType**.

Related files:

CreateTipperMamdaniExampleXML2.java TipperMamdani2.xml test-data-Tipper1.txt

**Mamdani3.**

This FLS is a modified version of Mamdani2. First of all, a third input (quality) is considered. It is implemented with the class **AggregatedFuzzyVariableType**. Moreover, it includes two linguistic terms implemented as **AggregatedFuzzyTermType**. The first term, acceptable, turns up as the combination food is delicious and service is good or excellent. Thus, it combines a t-norm (MIN) and a t-conorm (MAX). The second term, bad, comes out of evaluating food is rancid or service is poor.

Then, two additional rules are included in the rule base. They make use of the input variable quality.

Related files:

CreateTipperMamdaniExampleXML3.java TipperMamdani3.xml test-data-Tipper2.txt

**TSK.**

This FLS is very similar to Mamdani1 but now the output is defined as a **TskVariableType** and the 3 associated linguistic terms are implemented as both order-0 and order-1 **TskTermType**.

Related files:

CreateTipperTSKExampleXML.java TipperTSK.xml test-data-Tipper1.txt

**Java Fuzzy Markup Language. ©2019** 49

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

**Tsukamoto1.**

This FLS is very similar to Mamdani1 but now the output is defined as a **TsukamotoVariableType** and the 3 associated linguistic terms are implemented as monotone **TsukamotoTermType** membership functions.

Related files:

CreateTipperTsukamotoExampleXML1.java TipperTsukamoto1.xml test-data-Tipper1.txt

**Tsukamoto2.**

This FLS is dual to Tsukamoto1. The only difference arises from the fact that now the linguistic term cheap in the output variable tip is implemented with the class **PointSetMonotonicShapeType**. Moreover, we use CUBIC as **MonotonicInterpolationMethodType**.

Related files:

CreateTipperTsukamotoExampleXML2.java TipperTsukamoto2.xml test-data-Tipper1.txt

**AnYa.**

This FLS is very similar to Mamdani3 in the sense that it considers three inputs (food, service, and quality) but now each input is defined as an **AnYaDataCloudType** and the rule base is defined as an **AnYaRuleBaseType**.

Related files:

CreateTipperAnYaExampleXML.java TipperAnYa.xml test-data-Tipper2.txt

**Japanese Diet Assessment (Regression Problem)**

This problem is aimed at assessing the diet healthy level of japanese people who are expected to get a good balance between variety of food and drinks but also physical activity.

We have implemented a Mamdani-type FLS with 5 input variables: the Percentage of Calories from Carbohydrate (PCC), the Percentage of Calories

**Java Fuzzy Markup Language. ©2019** 50

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

from Protein (PCP), the Percentage of Calories from Fat (PCF), the Percentage of Caloric Ratio (PCR), and Food Group Balance (FGB). Each input is characterized by a strong fuzzy partition with 3 linguistic terms (low, medium, high) which are implemented as Trapezoidal (**FuzzyTermType.TYPE\_trapezoidShape**) membership functions. In addition, FLS has 1 regression output, the Dietary Healthy Level (DHL), which is characterized by a strong fuzzy partition with 5 linguistic terms (very low, low, medium, high, very high) which are implemented as Trapezoidal (**FuzzyTermType.TYPE\_trapezoidShape**) membership functions.

The rule base is implemented as **MamdaniRuleBaseType** and it is made up of just 2 rules. It applies the usual Mamdani min-max inference mechanism and the well-known **Center of Gravity** as defuzzification mechanism.

Related files:

CreateJapaneseDietAssessmentMamdaniExampleXML.java JapaneseDietAssessmentMamdani.xml test-data-JapaneseDietAssessment.txt

**Robot (Real-world Problem)**

This problem is aimed at producing the well-known wall-following behavior in robotics.

We took as starting point a fuzzy controller designed in accordance with the IEC 61131-7 standard (robot.fcl). Then, we have implemented a Mamdani-type FLS (RobotMamdani.xml) which comes out after using JFML to translate the given FCL file into an XML file which is fully compliant with the IEEE standard for FML.

The FLS is made up of 4 input variables: distances to the right (rd) and left walls (dq); orientation regarding to the wall (o); and velocity (V).

rd is described by a strong fuzzy partition with 4 linguistic terms (Low (L), Medium (M), High (H), Very High (VH)) which are implemented as Triangular (**FuzzyTermType.TYPE\_triangularShape**) membership functions. dq is described by a strong fuzzy partition with 2 linguistic terms (L, H) which are implemented as Triangular (**FuzzyTermType.TYPE\_triangularShape**) membership functions. o is described by a strong fuzzy partition with 5 linguistic terms (High left (HL), Low left (LL), zero (Z), Low right (LR), High right (HR)) which are implemented as Triangular (**FuzzyTermType.TYPE\_triangularShape**)

**Java Fuzzy Markup Language. ©2019** 51

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

membership functions. v is described by a strong fuzzy partition with 2 linguistic terms (L, H) which are implemented as Triangular (**FuzzyTermType.TYPE\_triangularShape**) membership functions.

It also has 2 output variables: linear acceleration (la) and angular velocity (av).

la is described by a strong fuzzy partition with 9 linguistic terms (VHB, HB, MB, SB, Z, SA, MA, HA, VHA) which are implemented as Triangular (**FuzzyTermType.TYPE\_triangularShape**) membership functions. av is described by a strong fuzzy partition with 9 linguistic terms (VHR, HR, MR, SR, Z, SL, ML, HL, VHL) which are implemented as Triangular (**FuzzyTermType.TYPE\_triangularShape**) membership functions.

Moreover, the rule base is implemented as **MamdaniRuleBaseType** and it is made up of 41 rules. It applies the usual Mamdani min-max inference mechanism and the well-known **Center of Gravity** as defuzzification mechanism.

Related files:

robot.fcl RobotMamdani.xml test-data-Robot.txt

**Users**

Here, we go in detail with 2 of the illustrative examples introduced above:

**Iris (Mamdani1) Inverted Pendulum (TSK1)**

For each example, we first explain how to create the related XML file and then how to evaluate the FLS.

**Iris (Mamdani1)**

Firstly, we create the Mamdani-type FLS (iris) with an empty KB (kb):

FuzzyInferenceSystem iris = new FuzzyInferenceSystem("iris - MAMDANI"); KnowledgeBaseType kb = new KnowledgeBaseType();

**Java Fuzzy Markup Language. ©2019** 52

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

iris.setKnowledgeBase(kb);

Then, we define and add to the KB all involved variables with their related linguistic terms. In this case, 1 input (pw) which is described by 3 linguistic terms (pw\_low, pw\_medium, pw\_high) implemented as Trapezoidal membership functions:

FuzzyVariableType pw = new FuzzyVariableType("PetalWidth", 0.1f, 2.5f); FuzzyTermType pw\_low = new FuzzyTermType("low", FuzzyTermType.TYPE\_trapezoidShape, (new float[] { 0.1f, 0.1f, 0.244f, 1.087f })); pw.addFuzzyTerm(pw\_low); FuzzyTermType pw\_medium = new FuzzyTermType("medium", FuzzyTermType.TYPE\_trapezoidShape, (new float[] { 0.244f, 1.087f, 1.419f, 1.906f })); pw.addFuzzyTerm(pw\_medium); FuzzyTermType pw\_high = new FuzzyTermType("high", FuzzyTermType.TYPE\_trapezoidShape, (new float[] { 1.419f, 1.906f, 2.5f, 2.5f })); pw.addFuzzyTerm(pw\_high); kb.addVariable(pw);

and 1 output (irisClass) which is described by 3 linguistic terms (irisClass\_setosa, irisClass\_virginica, irisClass\_versicolor) implemented as Singleton membership functions:

FuzzyVariableType irisClass = new FuzzyVariableType("irisClass", 1, 3); irisClass.setDefaultValue(1f); irisClass.setAccumulation("MAX"); irisClass.setDefuzzifierName("MOM"); irisClass.setType("output"); FuzzyTermType irisClass\_setosa = new FuzzyTermType("setosa", FuzzyTermType.TYPE\_singletonShape, (new float[] { 1f })); irisClass.addFuzzyTerm(irisClass\_setosa);

**Java Fuzzy Markup Language. ©2019** 53

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

FuzzyTermType irisClass\_virginica = new FuzzyTermType("virginica", FuzzyTermType.TYPE\_singletonShape, (new float[] { 2f })); irisClass.addFuzzyTerm(irisClass\_virginica); FuzzyTermType irisClass\_versicolor = new FuzzyTermType("versicolor", FuzzyTermType.TYPE\_singletonShape, (new float[] { 3f })); irisClass.addFuzzyTerm(irisClass\_versicolor); kb.addVariable(irisClass);

Notice that the definition of the output variable includes setting fuzzy operators: Maximum (MAX) for Accumulation and Mean of Max (MOM) for Defuzzifier, in this example.

Then, we create and add to KB the related rule base (rb) which in this case includes 3 rules (one per output class). Notice that we apply the usual min-max inference mechanism. Even though in this case there is only one input, it is mandatory to set the conective related to evaluation of rule antecedents. In this example, rule antecedents are combined by conective and which is implemented as the t-norm Minimum (MIN).

// Create an empty rule base MamdaniRuleBaseType rb = new MamdaniRuleBaseType("rulebase- iris");

// RULE 1 FuzzyRuleType r1 = new FuzzyRuleType("rule1", "and", "MIN", 1.0f); AntecedentType ant1 = new AntecedentType(); ant1.addClause(new ClauseType(pw, pw\_low)); ConsequentType con1 = new ConsequentType(); con1.addThenClause(irisClass, irisClass\_setosa); r1.setAntecedent(ant1); r1.setConsequent(con1); rb.addRule(r1);

// RULE 2 FuzzyRuleType r2 = new FuzzyRuleType("rule2", "and", "MIN", 1.0f);

**Java Fuzzy Markup Language. ©2019** 54

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

AntecedentType ant2 = new AntecedentType(); ant2.addClause(new ClauseType(pw, pw\_medium)); ConsequentType con2 = new ConsequentType(); con2.addThenClause(irisClass, irisClass\_virginica); r2.setAntecedent(ant2); r2.setConsequent(con2); rb.addRule(r2);

// RULE 3 FuzzyRuleType r3 = new FuzzyRuleType("rule3", "and", "MIN", 1.0f); AntecedentType ant3 = new AntecedentType(); ant3.addClause(new ClauseType(pw, pw\_high)); ConsequentType con3 = new ConsequentType(); con3.addThenClause(irisClass, irisClass\_versicolor); r3.setAntecedent(ant3); r3.setConsequent(con3); rb.addRule(r3);

// Add rb to the iris FLS iris.addRuleBase(rb);

The three rules created above are as follows:

r1: IF PetalWidth IS low THEN irisClass IS setosa r2: IF PetalWidth IS medium THEN irisClass IS virginica r3: IF PetalWidth IS high THEN irisClass IS versicolor

Once the FLS is created, it can be stored in an XML file:

File irisXMLFile = new File("./XMLFiles/IrisMamdani1.xml"); JFML.writeFSTtoXML(iris, irisXMLFile);

Then, it is time for evaluation.

The FLS can be directly evaluated through the command line with one specific data value assigned to each input variable:

**Java Fuzzy Markup Language. ©2019** 55

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

java -jar ./lib/JFML-vx.x.jar Iris Mamdani1 PetalWidth 0.2

1) Loading Fuzzy System from an XML file according the standard IEEE 1855

2) Setting input variables: PetalWidth=0.2

3) Making fuzzy inference RESULTS

(INPUT): PetalWidth=0.2 (OUTPUT): irisClass=1.0

4) Fuzzy System Description

FUZZY SYSTEM: iris - MAMDANI KNOWLEDGEBASE:

\*PetalWidth - domain[0.1, 2.5] - input

low - trapezoid [a: 0.1, b: 0.1, c: 0.244, d: 1.087]

medium - trapezoid [a: 0.244, b: 1.087, c: 1.419, d: 1.906]

high - trapezoid [a: 1.419, b: 1.906, c: 2.5, d: 2.5]

\*irisClass - domain[1.0, 3.0] - Accumulation:MAX; Defuzzifier:MOM - output

setosa - singleton [a: 1.0] virginica - singleton [a: 2.0] versicolor - singleton [a: 3.0]

RULEBASE:

\*mamdani - rulebase-iris: OR=MAX; AND=MIN; ACTIVATION=MIN

RULE 1: rule1 - (1.0) IF PetalWidth IS low THEN irisClass IS setosa [weight=1.0]

RULE 2: rule2 - (0.0) IF PetalWidth IS medium THEN irisClass IS virginica [weight=1.0]

RULE 3: rule3 - (0.0) IF PetalWidth IS high THEN irisClass IS versicolor [weight=1.0]

**Java Fuzzy Markup Language. ©2019** 56

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

or with one data file (which contains a data record for each row; 10 data records in this example):

java -jar ./lib/JFML-vx.x.jar Iris Mamdani1 ./XMLFiles/test-data-Iris1.txt

1) Loading Fuzzy System from an XML file according the standard IEEE 1855

2.0) Setting input variables: PetalWidth=0.2

3.0) Making fuzzy inference RESULTS

(INPUT): PetalWidth=0.2 (OUTPUT): irisClass=1.0

2.1) Setting input variables: PetalWidth=0.35

3.1) Making fuzzy inference RESULTS

(INPUT): PetalWidth=0.35 (OUTPUT): irisClass=1.0

2.2) Setting input variables: PetalWidth=0.55

3.2) Making fuzzy inference RESULTS

(INPUT): PetalWidth=0.55 (OUTPUT): irisClass=1.0

2.3) Setting input variables: PetalWidth=0.65

3.3) Making fuzzy inference RESULTS

(INPUT): PetalWidth=0.65 (OUTPUT): irisClass=1.0

2.4) Setting input variables: PetalWidth=0.82

3.4) Making fuzzy inference

**Java Fuzzy Markup Language. ©2019** 57

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

RESULTS

(INPUT): PetalWidth=0.82 (OUTPUT): irisClass=2.0

2.5) Setting input variables: PetalWidth=1.25

3.5) Making fuzzy inference RESULTS

(INPUT): PetalWidth=1.25 (OUTPUT): irisClass=2.0

2.6) Setting input variables: PetalWidth=1.54

3.6) Making fuzzy inference RESULTS

(INPUT): PetalWidth=1.54 (OUTPUT): irisClass=2.0

2.7) Setting input variables: PetalWidth=1.66

3.7) Making fuzzy inference RESULTS

(INPUT): PetalWidth=1.66 (OUTPUT): irisClass=2.0

2.8) Setting input variables: PetalWidth=1.75

3.8) Making fuzzy inference RESULTS

(INPUT): PetalWidth=1.75 (OUTPUT): irisClass=3.0

2.9) Setting input variables: PetalWidth=2.0

3.9) Making fuzzy inference RESULTS

(INPUT): PetalWidth=2.0 (OUTPUT): irisClass=3.0

4) Fuzzy System Description

**Java Fuzzy Markup Language. ©2019** 58

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

FUZZY SYSTEM: iris - MAMDANI KNOWLEDGEBASE:

\*PetalWidth - domain[0.1, 2.5] - input

low - trapezoid [a: 0.1, b: 0.1, c: 0.244, d: 1.087]

medium - trapezoid [a: 0.244, b: 1.087, c: 1.419, d: 1.906]

high - trapezoid [a: 1.419, b: 1.906, c: 2.5, d: 2.5]

\*irisClass - domain[1.0, 3.0] - Accumulation:MAX; Defuzzifier:MOM - output

setosa - singleton [a: 1.0] virginica - singleton [a: 2.0] versicolor - singleton [a: 3.0]

RULEBASE:

\*mamdani - rulebase-iris: OR=MAX; AND=MIN; ACTIVATION=MIN

RULE 1: rule1 - (0.0) IF PetalWidth IS low THEN irisClass IS setosa [weight=1.0]

RULE 2: rule2 - (0.0) IF PetalWidth IS medium THEN irisClass IS virginica [weight=1.0]

RULE 3: rule3 - (1.0) IF PetalWidth IS high THEN irisClass IS versicolor [weight=1.0]

**Inverted Pendulum (TSK1)**

Firstly, we create the TSK-type FLS (invertedPendulum) with an empty KB (kb):

FuzzyInferenceSystem invertedPendulum = new FuzzyInferenceSystem("invertedPendulum - TSK"); KnowledgeBaseType kb = new KnowledgeBaseType(); invertedPendulum.setKnowledgeBase(kb);

Then, we define and add to the KB all involved variables with their related linguistic terms. In this case, 2 inputs (ang and ca) which are described by 5 linguistic terms (vneg, neg, neu, pos, vpos) implemented as Trapezoidal and Triangular membership functions. In addition, we add 2 or composite linguistic

**Java Fuzzy Markup Language. ©2019** 59

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

terms to each input:

//FUZZY VARIABLE Angle FuzzyVariableType ang = new FuzzyVariableType("Angle", 0, 255);

// Basic linguistic terms FuzzyTermType ang\_vneg = new FuzzyTermType("very negative", FuzzyTermType.TYPE\_trapezoidShape, (new float[] { 0f, 0f, 48f, 88f })); ang.addFuzzyTerm(ang\_vneg); FuzzyTermType ang\_neg = new FuzzyTermType("negative", FuzzyTermType.TYPE\_triangularShape, (new float[] { 48f, 88f, 128f })); ang.addFuzzyTerm(ang\_neg); FuzzyTermType ang\_neu = new FuzzyTermType("zero", FuzzyTermType.TYPE\_triangularShape, (new float[] { 88f, 128f, 168f })); ang.addFuzzyTerm(ang\_neu); FuzzyTermType ang\_pos = new FuzzyTermType("positive", FuzzyTermType.TYPE\_triangularShape, (new float[] { 128f, 168f, 208f })); ang.addFuzzyTerm(ang\_pos); FuzzyTermType ang\_vpos = new FuzzyTermType("very positive", FuzzyTermType.TYPE\_trapezoidShape, (new float[] { 168f, 208f, 255f, 255f })); ang.addFuzzyTerm(ang\_vpos);

// OR composite linguistic terms OrLogicalType ang\_or1 = new OrLogicalType("BSUM", "very negative", "negative"); CircularDefinitionType ang\_c1 = new CircularDefinitionType(ang\_or1, ang); FuzzyTermType ang\_vneg\_or\_neg = new FuzzyTermType("very negative or negative", ang\_c1); ang.addFuzzyTerm(ang\_vneg\_or\_neg); OrLogicalType ang\_or2 = new OrLogicalType("BSUM", "positive", "very positive"); CircularDefinitionType ang\_c2 = new CircularDefinitionType(ang\_or2, ang);

**Java Fuzzy Markup Language. ©2019** 60

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

FuzzyTermType ang\_pos\_or\_vpos = new FuzzyTermType("positive or very positive", ang\_c2); ang.addFuzzyTerm(ang\_pos\_or\_vpos); kb.addVariable(ang);

//FUZZY VARIABLE ChangeAngle FuzzyVariableType ca = new FuzzyVariableType("ChangeAngle", 0, 255);

// Basic linguistic terms FuzzyTermType ca\_vneg = new FuzzyTermType("very negative", FuzzyTermType.TYPE\_trapezoidShape, (new float[] { 0f, 0f, 48f, 88f })); ca.addFuzzyTerm(ca\_vneg); FuzzyTermType ca\_neg = new FuzzyTermType("negative", FuzzyTermType.TYPE\_triangularShape, (new float[] { 48f, 88f, 128f })); ca.addFuzzyTerm(ca\_neg); FuzzyTermType ca\_neu = new FuzzyTermType("zero", FuzzyTermType.TYPE\_triangularShape, (new float[] { 88f, 128f, 168f })); ca.addFuzzyTerm(ca\_neu); FuzzyTermType ca\_pos = new FuzzyTermType("positive", FuzzyTermType.TYPE\_triangularShape, (new float[] { 128f, 168f, 208f })); ca.addFuzzyTerm(ca\_pos); FuzzyTermType ca\_vpos = new FuzzyTermType("very positive", FuzzyTermType.TYPE\_trapezoidShape, (new float[] { 168f, 208f, 255f, 255f })); ca.addFuzzyTerm(ca\_vpos);

// OR composite linguistic terms OrLogicalType ca\_or1 = new OrLogicalType("BSUM", "very negative", "negative"); CircularDefinitionType ca\_c1 = new CircularDefinitionType(ca\_or1, ca); FuzzyTermType ca\_vneg\_or\_neg = new FuzzyTermType("very negative or negative", ca\_c1); ca.addFuzzyTerm(ca\_vneg\_or\_neg); OrLogicalType ca\_or2 = new OrLogicalType("BSUM", "positive", "very positive");

**Java Fuzzy Markup Language. ©2019** 61

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

CircularDefinitionType ca\_c2 = new CircularDefinitionType(ca\_or2, ang); FuzzyTermType ca\_pos\_or\_vpos = new FuzzyTermType("positive or very positive", ca\_c2); ca.addFuzzyTerm(ca\_pos\_or\_vpos); kb.addVariable(ca);

and 1 output (force) which is described by 5 linguistic terms (force\_vneg, force\_neg, force\_neu, force\_pos, force\_vpos) implemented as order-0 TSK terms:

TskVariableType force = new TskVariableType("Force"); force.setDefaultValue(0f); force.setCombination("WA"); force.setType("output"); TskTermType force\_vneg = new TskTermType("very negative", TskTerm.\_ORDER\_0, (new float[] { 48f})); force.addTskTerm(force\_vneg); TskTermType force\_neg = new TskTermType("negative", TskTerm.\_ORDER\_0, (new float[] { 88f})); force.addTskTerm(force\_neg); TskTermType force\_neu = new TskTermType("zero", TskTerm.\_ORDER\_0, (new float[] { 128f})); force.addTskTerm(force\_neu); TskTermType force\_pos = new TskTermType("positive", TskTerm.\_ORDER\_0, (new float[] { 168f})); force.addTskTerm(force\_pos); TskTermType force\_vpos = new TskTermType("very positive", TskTerm.\_ORDER\_0, (new float[] { 208f})); force.addTskTerm(force\_vpos); kb.addVariable(force);

Notice that the definition of the output variable includes setting the combination method (Weighted Average (WA) in this example) as well as the default output value (zero in this example).

Then, we create and add to KB the related rule base (rb) which in this case includes 19 rules. In this example, rule antecedents are combined by conective

**Java Fuzzy Markup Language. ©2019** 62

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

and which is implemented as the t-norm Minimum (MIN).

// Create an empty rule base TskRuleBaseType rb = new TskRuleBaseType("rulebase1", FuzzySystemRuleBase.TYPE\_TSK); // RULE 1 TskFuzzyRuleType r1 = new TskFuzzyRuleType("rule1", "and", "MIN", 1.0f); AntecedentType ant1 = new AntecedentType(); ant1.addClause(new ClauseType(ang, ang\_vneg\_or\_neg)); ant1.addClause(new ClauseType(ca, ca\_vneg\_or\_neg)); TskConsequentType con1 = new TskConsequentType(); con1.addTskThenClause(force, force\_vneg); r1.setAntecedent(ant1); r1.setTskConsequent(con1); rb.addTskRule(r1); // RULE 2 TskFuzzyRuleType r2 = new TskFuzzyRuleType("rule2", "and", "MIN", 1.0f); AntecedentType ant2 = new AntecedentType(); ant2.addClause(new ClauseType(ang, ang\_vneg)); ant2.addClause(new ClauseType(ca, ca\_neu)); TskConsequentType con2 = new TskConsequentType(); con2.addTskThenClause(force, force\_vneg); r2.setAntecedent(ant2); r2.setTskConsequent(con2); rb.addTskRule(r2); // RULE 3 TskFuzzyRuleType r3 = new TskFuzzyRuleType("rule3", "and", "MIN", 1.0f); AntecedentType ant3 = new AntecedentType(); ant3.addClause(new ClauseType(ang, ang\_vneg)); ant3.addClause(new ClauseType(ca, ca\_pos)); TskConsequentType con3 = new TskConsequentType(); con3.addTskThenClause(force, force\_neg); r3.setAntecedent(ant3); r3.setTskConsequent(con3); rb.addTskRule(r3); // RULE 4 TskFuzzyRuleType r4 = new TskFuzzyRuleType("rule4", "and",

**Java Fuzzy Markup Language. ©2019** 63

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

"MIN", 1.0f); AntecedentType ant4 = new AntecedentType(); ant4.addClause(new ClauseType(ang, ang\_vneg)); ant4.addClause(new ClauseType(ca, ca\_vpos)); TskConsequentType con4 = new TskConsequentType(); con4.addTskThenClause(force, force\_neu); r4.setAntecedent(ant4); r4.setTskConsequent(con4); rb.addTskRule(r4); // RULE 5 TskFuzzyRuleType r5 = new TskFuzzyRuleType("rule5", "and", "MIN", 1.0f); AntecedentType ant5 = new AntecedentType(); ant5.addClause(new ClauseType(ang, ang\_neg)); ant5.addClause(new ClauseType(ca, ca\_neu)); TskConsequentType con5 = new TskConsequentType(); con5.addTskThenClause(force, force\_neg); r5.setAntecedent(ant5); r5.setTskConsequent(con5); rb.addTskRule(r5); // RULE 6 TskFuzzyRuleType r6 = new TskFuzzyRuleType("rule6", "and", "MIN", 1.0f); AntecedentType ant6 = new AntecedentType(); ant6.addClause(new ClauseType(ang, ang\_neg)); ant6.addClause(new ClauseType(ca, ca\_pos)); TskConsequentType con6 = new TskConsequentType(); con6.addTskThenClause(force, force\_neu); r6.setAntecedent(ant6); r6.setTskConsequent(con6); rb.addTskRule(r6); // RULE 7 TskFuzzyRuleType r7 = new TskFuzzyRuleType("rule7", "and", "MIN", 1.0f); AntecedentType ant7 = new AntecedentType(); ant7.addClause(new ClauseType(ang, ang\_neg)); ant7.addClause(new ClauseType(ca, ca\_vpos)); TskConsequentType con7 = new TskConsequentType(); con7.addTskThenClause(force, force\_pos); r7.setAntecedent(ant7); r7.setTskConsequent(con7);

**Java Fuzzy Markup Language. ©2019** 64

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

rb.addTskRule(r7); // RULE 8 TskFuzzyRuleType r8 = new TskFuzzyRuleType("rule8", "and", "MIN", 1.0f); AntecedentType ant8 = new AntecedentType(); ant8.addClause(new ClauseType(ang, ang\_neu)); ant8.addClause(new ClauseType(ca, ca\_vneg)); TskConsequentType con8 = new TskConsequentType(); con8.addTskThenClause(force, force\_vneg); r8.setAntecedent(ant8); r8.setTskConsequent(con8); rb.addTskRule(r8); // RULE 9 TskFuzzyRuleType r9 = new TskFuzzyRuleType("rule9", "and", "MIN", 1.0f); AntecedentType ant9 = new AntecedentType(); ant9.addClause(new ClauseType(ang, ang\_neu)); ant9.addClause(new ClauseType(ca, ca\_neg)); TskConsequentType con9 = new TskConsequentType(); con9.addTskThenClause(force, force\_neg); r9.setAntecedent(ant9); r9.setTskConsequent(con9); rb.addTskRule(r9); // RULE 10 TskFuzzyRuleType r10 = new TskFuzzyRuleType("rule10", "and", "MIN", 1.0f); AntecedentType ant10 = new AntecedentType(); ant10.addClause(new ClauseType(ang, ang\_neu)); ant10.addClause(new ClauseType(ca, ca\_neu)); TskConsequentType con10 = new TskConsequentType(); con10.addTskThenClause(force, force\_neu); r10.setAntecedent(ant10); r10.setTskConsequent(con10); rb.addTskRule(r10); // RULE 11 TskFuzzyRuleType r11 = new TskFuzzyRuleType("rule11", "and", "MIN", 1.0f); AntecedentType ant11 = new AntecedentType(); ant11.addClause(new ClauseType(ang, ang\_neu)); ant11.addClause(new ClauseType(ca, ca\_pos)); TskConsequentType con11 = new TskConsequentType();

**Java Fuzzy Markup Language. ©2019** 65

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

con11.addTskThenClause(force, force\_pos); r11.setAntecedent(ant11); r11.setTskConsequent(con11); rb.addTskRule(r11); // RULE 12 TskFuzzyRuleType r12 = new TskFuzzyRuleType("rule12", "and", "MIN", 1.0f); AntecedentType ant12 = new AntecedentType(); ant12.addClause(new ClauseType(ang, ang\_neu)); ant12.addClause(new ClauseType(ca, ca\_vpos)); TskConsequentType con12 = new TskConsequentType(); con12.addTskThenClause(force, force\_vpos); r12.setAntecedent(ant12); r12.setTskConsequent(con12); rb.addTskRule(r12); // RULE 13 TskFuzzyRuleType r13 = new TskFuzzyRuleType("rule13", "and", "MIN", 1.0f); AntecedentType ant13 = new AntecedentType(); ant13.addClause(new ClauseType(ang, ang\_pos)); ant13.addClause(new ClauseType(ca, ca\_vneg)); TskConsequentType con13 = new TskConsequentType(); con13.addTskThenClause(force, force\_neg); r13.setAntecedent(ant13); r13.setTskConsequent(con13); rb.addTskRule(r13); // RULE 14 TskFuzzyRuleType r14 = new TskFuzzyRuleType("rule14", "and", "MIN", 1.0f); AntecedentType ant14 = new AntecedentType(); ant14.addClause(new ClauseType(ang, ang\_pos)); ant14.addClause(new ClauseType(ca, ca\_neg)); TskConsequentType con14 = new TskConsequentType(); con14.addTskThenClause(force, force\_neu); r14.setAntecedent(ant14); r14.setTskConsequent(con14); rb.addTskRule(r14); // RULE 15 TskFuzzyRuleType r15 = new TskFuzzyRuleType("rule15", "and", "MIN", 1.0f); AntecedentType ant15 = new AntecedentType();

**Java Fuzzy Markup Language. ©2019** 66

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

ant15.addClause(new ClauseType(ang, ang\_pos)); ant15.addClause(new ClauseType(ca, ca\_neu)); TskConsequentType con15 = new TskConsequentType(); con15.addTskThenClause(force, force\_pos); r15.setAntecedent(ant15); r15.setTskConsequent(con15); rb.addTskRule(r15); // RULE 16 TskFuzzyRuleType r16 = new TskFuzzyRuleType("rule16", "and", "MIN", 1.0f); AntecedentType ant16 = new AntecedentType(); ant16.addClause(new ClauseType(ang, ang\_vpos)); ant16.addClause(new ClauseType(ca, ca\_vneg)); TskConsequentType con16 = new TskConsequentType(); con16.addTskThenClause(force, force\_neu); r16.setAntecedent(ant16); r16.setTskConsequent(con16); rb.addTskRule(r16); // RULE 17 TskFuzzyRuleType r17 = new TskFuzzyRuleType("rule17", "and", "MIN", 1.0f); AntecedentType ant17 = new AntecedentType(); ant17.addClause(new ClauseType(ang, ang\_vpos)); ant17.addClause(new ClauseType(ca, ca\_neg)); TskConsequentType con17 = new TskConsequentType(); con17.addTskThenClause(force, force\_pos); r17.setAntecedent(ant17); r17.setTskConsequent(con17); rb.addTskRule(r17); // RULE 18 TskFuzzyRuleType r18 = new TskFuzzyRuleType("rule18", "and", "MIN", 1.0f); AntecedentType ant18 = new AntecedentType(); ant18.addClause(new ClauseType(ang, ang\_vpos)); ant18.addClause(new ClauseType(ca, ca\_neu)); TskConsequentType con18 = new TskConsequentType(); con18.addTskThenClause(force, force\_vpos); r18.setAntecedent(ant18); r18.setTskConsequent(con18); rb.addTskRule(r18); // RULE 19

**Java Fuzzy Markup Language. ©2019** 67

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

TskFuzzyRuleType r19 = new TskFuzzyRuleType("rule19", "and", "MIN", 1.0f); AntecedentType ant19 = new AntecedentType(); ant19.addClause(new ClauseType(ang, ang\_pos\_or\_vpos)); ant19.addClause(new ClauseType(ca, ca\_pos\_or\_vpos)); TskConsequentType con19 = new TskConsequentType(); con19.addTskThenClause(force, force\_vpos); r19.setAntecedent(ant19); r19.setTskConsequent(con19); rb.addTskRule(r19); // Add rb to the invertedPendulum FLS invertedPendulum.addRuleBase(rb);

The rules created above are as follows:

r1: IF Angle IS very negative or negative AND ChangeAngle IS very negative or negative THEN Force IS very negative r2: IF Angle IS very negative AND ChangeAngle IS zero THEN Force IS very negative r3: IF Angle IS very negative AND ChangeAngle IS positive THEN Force IS negative r4: IF Angle IS very negative AND ChangeAngle IS very positive THEN Force IS zero r5: IF Angle IS negative AND ChangeAngle IS zero THEN Force IS negative r6: IF Angle IS negative AND ChangeAngle IS positive THEN Force IS zero r7: IF Angle IS negative AND ChangeAngle IS very positive THEN Force IS positive r8: IF Angle IS zero AND ChangeAngle IS very negative THEN Force IS very negative r9: IF Angle IS zero AND ChangeAngle IS negative THEN Force IS negative r10: IF Angle IS zero AND ChangeAngle IS zero THEN Force IS zero r11: IF Angle IS zero AND ChangeAngle IS positive THEN Force IS positive r12: IF Angle IS zero AND ChangeAngle IS very positive THEN Force IS very positive r13: IF Angle IS positive AND ChangeAngle IS very negative THEN Force IS negative

**Java Fuzzy Markup Language. ©2019** 68

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

r14: IF Angle IS positive AND ChangeAngle IS negative THEN Force IS zero r15: IF Angle IS positive AND ChangeAngle IS zero THEN Force IS positive r16: IF Angle IS very positive AND ChangeAngle IS very negative THEN Force IS zero r17: IF Angle IS very positive AND ChangeAngle IS negative THEN Force IS positive r18: IF Angle IS very positive AND ChangeAngle IS zero THEN Force IS very positive r19: IF Angle IS positive or very positive AND ChangeAngle IS positive or very positive THEN Force IS very positive

Once the FLS is created, it can be stored in an XML file:

File invertedPendulumXMLFile = new File("./XMLFiles/InvertedPendulumTSK1.xml"); JFML.writeFSTtoXML(invertedPendulum, invertedPendulumXMLFile);

Then, it is time for evaluation.

The FLS can be directly evaluated through the command line with one specific data value assigned to each input variable (see the example given above for iris - Mamdani1), or with one data file (which contains a data record for each row; 3 data records in this example):

java -jar ./lib/JFML-vx.x.jar InvertedPendulum TSK1 ./XMLFiles/test-data-InvertedPendulum.txt

1) Loading Fuzzy System from an XML file according the standard IEEE 1855

2.0) Setting input variables: Angle=10.0, ChangeAngle=10.0

3.0) Making fuzzy inference RESULTS

(INPUT): Angle=10.0, ChangeAngle=10.0

**Java Fuzzy Markup Language. ©2019** 69

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

(OUTPUT): Force=48.0

2.1) Setting input variables: Angle=125.0, ChangeAngle=125.0

3.1) Making fuzzy inference RESULTS

(INPUT): Angle=125.0, ChangeAngle=125.0 (OUTPUT): Force=117.565216

2.2) Setting input variables: Angle=250.0, ChangeAngle=250.0

3.2) Making fuzzy inference RESULTS

(INPUT): Angle=250.0, ChangeAngle=250.0 (OUTPUT): Force=208.0

4) Fuzzy System Description

FUZZY SYSTEM: invertedPendulum - TSK KNOWLEDGEBASE:

\*Angle - domain[0.0, 255.0] - input

very negative - trapezoid [a: 0.0, b: 0.0, c: 48.0, d: 88.0]

negative - triangular [a: 48.0, b: 88.0, c: 128.0] zero - triangular [a: 88.0, b: 128.0, c: 168.0] positive - triangular [a: 128.0, b: 168.0, c: 208.0]

very positive - trapezoid [a: 168.0, b: 208.0, c: 255.0, d: 255.0]

very negative or negative - very negative OR negative

positive or very positive - positive OR very positive

\*ChangeAngle - domain[0.0, 255.0] - input

very negative - trapezoid [a: 0.0, b: 0.0 , c: 48.0, d: 88.0]

negative - triangular [a: 48.0, b: 88.0, c: 128.0] zero - triangular [a: 88.0, b: 128.0, c: 168.0]

**Java Fuzzy Markup Language. ©2019** 70

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

positive - triangular [a: 128.0, b: 168.0, c: 208.0]

very positive - trapezoid [a: 168.0, b: 208.0, c: 255.0, d: 255.0]

very negative or negative - very negative OR negative

positive or very positive - positive OR very positive

\*Force - output

very negative - z = 48.0 negative - z = 88.0 zero - z = 128.0 positive - z = 168.0 very positive - z = 208.0

RULEBASE:

\*tsk - rulebase1: OR=MAX; AND=MIN; ACTIVATION=MIN

RULE 1: rule1 - (0.0) IF Angle IS very negative or negative AND ChangeAngle IS very negative or negative THEN Force IS very negative [weight=1.0]

RULE 2: rule2 - (0.0) IF Angle IS very negative AND ChangeAngle IS zero THEN Force IS very negative [weight=1.0]

RULE 3: rule3 - (0.0) IF Angle IS very negative AND ChangeAngle IS positive THEN Force IS negative [weight=1.0] RULE 4: rule4 - (0.0) IF Angle IS very negative AND ChangeAngle IS very positive THEN Force IS zero [weight=1.0]

RULE 5: rule5 - (0.0) IF Angle IS negative AND ChangeAngle IS zero THEN Force IS negative [weight=1.0]

RULE 6: rule6 - (0.0) IF Angle IS negative AND ChangeAngle IS positive THEN Force IS zero [weight=1.0]

RULE 7: rule7 - (0.0) IF Angle IS negative AND ChangeAngle IS very positive THEN Force IS positive [weight=1.0]

RULE 8: rule8 - (0.0) IF Angle IS zero AND ChangeAngle IS very negative THEN Force IS very negative [weight=1.0]

RULE 9: rule9 - (0.0) IF Angle IS zero AND ChangeAngle IS negative THE N Force IS negative [weight=1.0]

**Java Fuzzy Markup Language. ©2019** 71

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

RULE 10: rule10 - (0.0) IF Angle IS zero AND ChangeAngle IS zero THEN Force IS zero [weight=1.0]

RULE 11: rule11 - (0.0) IF Angle IS zero AND ChangeAngle IS positive THEN Force IS positive [weight=1.0]

RULE 12: rule12 - (0.0) IF Angle IS zero AND ChangeAngle IS very positive THEN Force IS very positive [weight=1.0]

RULE 13: rule13 - (0.0) IF Angle IS positive AND ChangeAngle IS very negative THEN Force IS negative [weight=1.0]

RULE 14: rule14 - (0.0) IF Angle IS positive AND ChangeAngle IS negative THEN Force IS zero [weight=1.0]

RULE 15: rule15 - (0.0) IF Angle IS positive AND ChangeAngle IS zero THEN Force IS positive [weight=1.0]

RULE 16: rule16 - (0.0) IF Angle IS very positive AND ChangeAngle IS very negative THEN Force IS zero [weight=1.0]

RULE 17: rule17 - (0.0) IF Angle IS very positive AND ChangeAngle IS negative THEN Force IS positive [weight=1.0]

RULE 18: rule18 - (0.0) IF Angle IS very positive AND ChangeA ngle IS zero THEN Force IS very positive [weight=1.0]

RULE 19: rule19 - (1.0) IF Angle IS positive or very positive AND ChangeAngle IS positive or very positive THEN Force IS very positive [weight=1.0]

**Developers**

Here, we provide developers with some details about programming code for 2 of the illustrative examples introduced above:

**Japanese Diet Assessment Tipper (Mamdani3)**

**Japanese Diet Assessment**

In this example, we would like to go in depth about how to make FLS evaluation from the point of view of developers. Notice that the class EvaluateExample.java in the package jfml.test is ready to help users to test all examples provided in this manual. However, in case of creating a new FLS system, developers may

**Java Fuzzy Markup Language. ©2019** 72

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

desire to develop a specific evaluation class for it.

Let us suppose that we want to create a new class EvaluateJapaneseDietAssessment.java for evaluating the FLS in /XMLFiles/JapaneseDietAssessmentMamdani.xml. First of all, we need to read the XML file and load the related FLS.

File xml = new File("./XMLFiles/JapaneseDietAssessmentMamdani.xml"); //loading FLS from an XML file according to the standard IEEE 1855 FuzzyInferenceSystem fs = JFML.load(xml);

Then, we have to set the input data values to test fs.

KnowledgeBaseVariable input1 = fs.getVariable("PCC"); KnowledgeBaseVariable input2 = fs.getVariable("PCP"); KnowledgeBaseVariable input3 = fs.getVariable("PCF"); KnowledgeBaseVariable input4 = fs.getVariable("PCR"); KnowledgeBaseVariable input5 = fs.getVariable("FGB"); input1.setValue(45.0); input2.setValue(21.6); input3.setValue(31.1); input4.setValue(82.9); input5.setValue(3.95);

Notice that the class FuzzyInferenceSystem includes the method evaluate() which is ready to run the selected inference mechanism for the given input data.

fs.evaluate();

Then, results of inference can be read and print as follows:

// get inferred output KnowledgeBaseVariable output = fs.getVariable("DHL"); float value = output.getValue();

**Java Fuzzy Markup Language. ©2019** 73

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

//Printing results System.out.println("RESULTS"); System.out.println(" (INPUT1): "+input1.getName()+ "="+input1.getValue()); System.out.println(" (INPUT2): "+input2.getName()+ "="+input2.getValue()); System.out.println(" (INPUT3): "+input3.getName()+ "="+input3.getValue()); System.out.println(" (INPUT4): "+input4.getName()+ "="+input4.getValue()); System.out.println(" (INPUT5): "+input5.getName()+ "="+input5.getValue()); System.out.println(" (OUTPUT): "+output.getName()+"="+ value); //Printing textual description of the FuzzySystem System.out.println(fs.toString());

**Tipper (Mamdani3)**

This example shows how to create an **aggregatedFuzzyVariable**. Namely, the input variable quality is defined as the aggregation of the other 2 inputs (food and service).

AggregatedFuzzyVariableType quality = new AggregatedFuzzyVariableType("quality");

// AGGREGATED FUZZY TERM acceptable AggregatedFuzzyTermType acceptable = new AggregatedFuzzyTermType("acceptable"); ClauseType acceptable\_t1 = new ClauseType(food,delicious); ClauseType acceptable\_t2 = new ClauseType(service,good); ClauseType acceptable\_t3 = new ClauseType(service,excellent); OrAggregatedType acceptable\_or = new OrAggregatedType(acceptable\_t2, acceptable\_t3); AndAggregatedType acceptable\_and = new AndAggregatedType(acceptable\_t1, acceptable\_or); acceptable.setAnd(acceptable\_and);

**Java Fuzzy Markup Language. ©2019** 74

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

// AGGREGATED FUZZY TERM bad AggregatedFuzzyTermType bad = new AggregatedFuzzyTermType("bad"); ClauseType bad\_t1 = new ClauseType(food,rancid); ClauseType bad\_t2 = new ClauseType(service,poor); OrAggregatedType bad\_or = new OrAggregatedType(bad\_t1, bad\_t2); bad.setOr(bad\_or); quality.addAggregatedFuzzyTerm(acceptable); quality.addAggregatedFuzzyTerm(bad); kb.addVariable(quality);

Aggregation is made, by default, using the t-norm MIN in the case of **AndAggregatedType** and the t-conorm MAX in the case of **OrAggregatedType**. Of course, the rest of t-norms (Nilpotent minimum, Bounded difference, Product, Drastic product, Einstein product, Hamacher product) and t-conorms (Nilpotent maximum, Probabilistic sum, Bounded sum, Drastic sum, Einstein sum, Hamacher sum) described in the IEEE Std 1855 can be used. Moreover, developers can implement their own t-norms and t-conorms. To do so, they have to overwrite the related custom methods:

private float custom\_and(float x, float y, String act) {

// TODO return 0; }private float custom\_or(float x, float y, String orMethod) {

// TODO return 0; }In addition, aggregated variables can be used in the rule base like the other variables. For instance, let us create a couple of rules:

// RULE 4 FuzzyRuleType rule4 = new FuzzyRuleType("rule4", "or", "MAX", 1.0f);

**Java Fuzzy Markup Language. ©2019** 75

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

AntecedentType ant4 = new AntecedentType(); ant4.addClause(new ClauseType(quality, acceptable)); ConsequentType con4 = new ConsequentType(); con4.addThenClause(tip, generous); rule4.setAntecedent(ant4); rule4.setConsequent(con4); rb.addRule(rule4);

// RULE 5 FuzzyRuleType rule5 = new FuzzyRuleType("rule5", "or", "MAX", 1.0f); AntecedentType ant5 = new AntecedentType(); ant5.addClause(new ClauseType(quality, bad, "very")); ConsequentType con5 = new ConsequentType(); con5.addThenClause(tip, cheap); rule5.setAntecedent(ant5); rule5.setConsequent(con5); rb.addRule(rule5);

The two rules created above are as follows:

r4: IF quality IS acceptable THEN tip IS generous r5: IF quality IS very bad THEN tip IS cheap

Notice that we also use the linguistic modifier very in the antecedent of rule r5.

**Embedded System examples**

We illustrate the potential of the JFML module for embedded systems with some real world problems.

**Home-made mobile robot for wall-following Ventilation system for a refrigerating chamber**

**Home-made mobile robot for wall-following**

We have developed a home-made mobile robot with low-cost components for this example. The architecture of this robot consists of an Arduino MEGA 2560 connected via USB with a Raspberry Pi, five ultrasonic sensors HC-SR04, one accelerometer/gyroscope sensor MPU6050, a servo SG-90 and two DC drive motors with matching wheels and driver H-Bridge L298N.

**Java Fuzzy Markup Language. ©2019** 76

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

Hardware architecture and a photography of our home-made mobile robot:

**Java Fuzzy Markup Language. ©2019** 77

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

In this case, a Raspberry Pi is used as a remote computer where the fuzzy inference is performed by taking the input values (from ultrasonic and accelerometer/gyroscope sensors) which are connected to the Arduino board. In other words, the FLC is embedded in the Arduino board but the fuzzy inference is carried out in the Raspberry Pi (where JFML is installed).

Notice that the sensor values are not directly used as inputs of the FLC (RobotIEEEstd1855.xml). They are low-level input variables that do not provide by themselves information that is relevant and meaningful to the FLC. For this reason, these low-level values are aggregated in JFML by means of the AggregatedSensor class in order to generate high-level input variables for the FLC. Likewise, the FLC outputs are sent to other aggregated sensors to calculate the values that will be sent to the DC motors and the servo for the linear acceleration and the angular velocity respectively.

The necessary steps to run the FLC with the Embedded System module are as follows. The first step is to read the description of the FLC from the related FML/XML document.

File fml=new File("./Examples/XMLFiles/RobotIEEEstd1855.xml"); FuzzyInferenceSystem fs = JFML.load(fml);

The second step is to instantiate the sensors/actuators that are connected with the Arduino board.

KnowledgeBaseVariable rd = fs.getVariable("rd"); KnowledgeBaseVariable dq = fs.getVariable("dq"); KnowledgeBaseVariable o = fs.getVariable("o"); KnowledgeBaseVariable v = fs.getVariable("v"); KnowledgeBaseVariable la = fs.getVariable("la"); KnowledgeBaseVariable av = fs.getVariable("av");

Sensor rdSF = new ArduinoHC\_SR04(rd.getName()+"front", ArduinoPin.PIN\_40, ArduinoPin.PIN\_41, 10, 200, true, 3, true, true); Sensor rdSF2 = new ArduinoHC\_SR04(rd.getName()+"front2", ArduinoPin.PIN\_48, ArduinoPin.PIN\_49, true, 3, false,true); Sensor rdSR = new ArduinoHC\_SR04(rd.getName()+"right",

**Java Fuzzy Markup Language. ©2019** 78

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

ArduinoPin.PIN\_50, ArduinoPin.PIN\_51, true, 3, false,true); Sensor rdSR2 = new ArduinoHC\_SR04(rd.getName()+"right2", ArduinoPin.PIN\_32, ArduinoPin.PIN\_33, true, 3,false, true); Sensor dqSL = new ArduinoHC\_SR04(dq.getName()+"left", ArduinoPin.PIN\_30, ArduinoPin.PIN\_31, true, 3, false,true); Sensor oS = new ArduinoMPU6050(o.getName(), -45, 45, true); Sensor laS = new ArduinoH\_BRIDGE\_L298N(la.getName(), ArduinoPin.PIN\_6, ArduinoPin.PIN\_7, ArduinoPin.PIN\_4, ArduinoPin.PIN\_2, ArduinoPin.PIN\_1, ArduinoPin.PIN\_5, -1, 1, 40, 70, 15); Sensor avS = new ArduinoSERVO(av.getName(), ArduinoPin.PIN\_9, -1, 1, 45, 135, 45, true);

The third step is to generate the aggregated sensors for the input and output values.

ArrayList sensorsRD = new ArrayList<>(); sensorsRD.add(rdSF); sensorsRD.add(rdSF2); sensorsRD.add(rdSR); sensorsRD.add(rdSR2);

AggregatedSensor rdAgg, dqAgg, oAgg, avAgg, vAgg; ArrayList sensorsDQ, sensorsO, sensorsAV;

rdAgg = new ArduinoAggregatedSensorRD(rd.getName(), sensorsRD, 0, 3, 6, 50, true); sensorsDQ = new ArrayList<>(); sensorsDQ.add(dqSL); dqAgg = new ArduinoAggregatedSensorDQ( dq.getName(), sensorsDQ, 0, 2);

sensorsO = new ArrayList<>(); sensorsO.add(rdSR); sensorsO.add(rdSR2); oAgg = new ArduinoAggregatedSensorO(o.getName(), sensorsO, -45, 45); sensorsAV = new ArrayList<>(); sensorsAV.add(avS);

**Java Fuzzy Markup Language. ©2019** 79

JFML (Java Fuzzy Markup Language) - Documentation (v1.2.4)

vAgg = new ArduinoAggregatedSensorV(v.getName(), -1, 1, 50); avAgg = new ArduinoAggregatedSensorAV(av.getName(), sensorsAV, -1, 1, 18);

The fourth step is to map the input/output variables with the sensors/actuators and to include them in a list.

ArrayList eVar = new ArrayList<>(); eVar.add(new EmbeddedVariableArduino(0,rd,rdSF)); eVar.add(new EmbeddedVariableArduino(0,rd,rdSF2)); eVar.add(new EmbeddedVariableArduino(0,rd,rdSR)); eVar.add(new EmbeddedVariableArduino(0,rd,rdSR2)); eVar.add(new EmbeddedVariableArduino(0,rd,rdAgg)); eVar.add(new EmbeddedVariableArduino(1,dq,dqSL)); eVar.add(new EmbeddedVariableArduino(1,dq,dqAgg)); eVar.add(new EmbeddedVariableArduino(2,o,oS)); eVar.add(new EmbeddedVariableArduino(2,o,oAgg)); eVar.add(new EmbeddedVariableArduino(3,v,vAgg)); eVar.add(new EmbeddedVariableArduino(4,la,laS)); eVar.add(new EmbeddedVariableArduino(5,av,avS)); eVar.add(new EmbeddedVariableArduino(5,av,avAgg));

The fifth step is to create the embedded system (including the name, port, rate, etc.)

EmbeddedSystem robot = new EmbeddedSystemArduinoUSB("Robot","USB\_PORT", 9600, eVar);

An .ino file can be automatically created and ready to be written in the Arduino board with the IDE provided by the company, and then the board can be connected to the Raspberry through a USB connection.

robot.createRunnableEmbeddedFile("RobotIEEEstd1855\_Arduino. ino");

**Java Fuzzy Markup Language. ©2019** 80