



Defence Research and  
Development Canada

Recherche et développement  
pour la défense Canada



# Multi-reasoner Inference

## *Software Interface Design Description*

Guillaume Morin-Brassard, Vincent Giroux  
*Fujitsu Consulting (Canada) Inc.*

The scientific or technical validity of this contract report is entirely the responsibility of the contractor and the contents do not necessarily have the approval or endorsement of the Department of National Defence of Canada.

**Defence Research and Development Canada – Valcartier**

Contract Report  
DRDC Valcartier CR 2012-003  
January 2012

**Canada**



# **Multi-reasoner Inference**

## *Software Interface Design Description*

Guillaume Morin-Brassard, Vincent Giroux  
*Fujitsu Consulting (Canada) Inc.*

Prepared by:  
Fujitsu Consulting (Canada) Inc.  
2000 Boulevard Lebourgneuf,  
Bureau 300, Québec (Québec) G2K 0E8

Contractor's document number: MRI-242-0449  
Contract project manager: Gilles Clairoux, 514-393-8822 x318  
PWGSC contract number: W7701-10-4064  
CSA: Étienne Martineau, Defense scientist, 418-844-4000 x4501

The scientific or technical validity of this contract report is entirely the responsibility of the contractor and the contents do not necessarily have the approval or endorsement of the Department of National Defence of Canada.

## **Defence Research and Development Canada – Valcartier**

Contract Report  
DRDC Valcartier CR 2012-003  
January 2012

## **IMPORTANT INFORMATIVE STATEMENTS**

The scientific or technical validity of this contract report is entirely the responsibility of the contractor and the contents do not necessarily have the approval or endorsement of the Department of National Defence of Canada.

- © Her Majesty the Queen in Right of Canada, as represented by the Minister of National Defence, 2012.
- © Sa Majesté la Reine (en droit du Canada), telle que représentée par le ministre de la Défense nationale, 2012.

## Abstract

---

To support its research activities in the intelligence domain, the Intelligence and Information (I&I) Section at DRDC Valcartier is developing the Intelligence Science & Technology Platform (ISTIP) as a major component of its R&D infrastructures. To improve the reasoning capabilities of the platform, the mandate of this contract is to produce a Multi-Reasoner Inference (MRI) capability based on the Multi-Intelligence Tool Suite (MITS) and the ISTIP software components previously developed by the I&I Section. Five main different services have been developed containing four individual reasoners and one multi-reasoner orchestrator. The reasoners that have been created are a Case-Based Reasoner (CBR), a Rule-Based Reasoner (RBR), a Descriptive-Logic Reasoner (DLR) and a KInematics and Geospatial Analysis Reasoner (KIGAR) which is based on the KIGAM module of the Inference of Situational Facts through Automated Reasoning (ISFAR) tool. Through the use of a common reasoning framework, these reasoners can now leverage their reasoning capabilities by sharing their strength to other reasoners and achieve an amazing synergy. This document describes the Software Interface Design Description of the MRI.

## Résumé

---

Afin de supporter ces activités de recherche dans le domaine du renseignement, la Section du Renseignement et Information de RDDC Valcartier développe la Plate-forme de Science et Technologie du Renseignement (ISTIP) comme un composant majeur de ses infrastructures de R&D. Afin d'améliorer les aptitudes de raisonnement de la plate-forme, le mandat de ce contrat est de créer un outil d'inférence Multi-Raisonneur (MRI) basé sur la « Multi-Intelligence Tool Suite » (MITS) et sur les composants logiciels déjà implémentés par la section I&I. Cinq différents services ont été développés comprenant quatre raisonneurs individuels et un orchestrateur multi-raisonneur. Les raisonneurs qui ont été créés sont un raisonneur par cas (CBR), un raisonneur par règles (RBR), un raisonneur ontologique (DLR) et un raisonneur d'analyse cinématique et géo-spatiale (KIGAR) basé sur le module KIGAM de l'outil d'Inférence Automatisée de Faits Situationnels (ISFAR). Grâce à l'utilisation d'un cadre de raisonnement commun, ces raisonneurs peuvent désormais exploiter leurs capacités de raisonnement en partageant leurs forces à d'autres raisonneurs et parvenir à une synergie épatante. Ce document décrit l'Architecture Logicielle des Interfaces du MRI.

This page intentionally left blank.

## Executive summary

---

### Multi-reasoner Inference: Software Interface Design Description

**Guillaume Morin-Brassard; Vincent Giroux; DRDC Valcartier CR 2012-003;  
Defence Research and Development Canada – Valcartier; January 2012.**

**Introduction or background:** To support its research activities in the intelligence domain, the Intelligence and Information (I&I) Section at DRDC Valcartier is developing the Intelligence Science & Technology Platform (ISTIP) as a major component of its R&D infrastructures. To improve the reasoning capabilities of the platform, the mandate of this contract is to produce a Multi-Reasoner Inference (MRI) capability based on the Multi-Intelligence Tool Suite (MITS) and the ISTIP software components previously developed by the I&I Section.

This document presents the Software Interface Design Description (SIDD) for the Multi-Reasoner Inference service and related reasoners services, according to the IEEE 12207.

Its purpose is to identify and describe interfaces to other systems, subsystems and applications. The interfaces identified and described should also be reflected in the Software Architecture Description. The Software Interface Design Description is the basis for detailed data and process design for interfaces. Describing the interfaces exposed by the Multi-Reasoners Inference services (including each individual reasoner and the MRI orchestrator), the call workflow and the necessary objects used to communicate with the services.

This document does not include the internal interfaces used within the MRI orchestrator or within the different reasoner services and the other sub-components.

**Results:** Five main different services have been developed containing four individual reasoners and one multi-reasoner orchestrator. The reasoners that have been created are a Case-Based Reasoner

(CBR), a Rule-Based Reasoner (RBR), a Descriptive-Logic Reasoner (DLR) and a Kinematics and Geospatial Analysis Reasoner (KIGAR) which is based on the KIGAM module of the Inference of Situational Facts Through Automated Reasoning (ISFAR) tool.

# Sommaire

---

## Multi-reasoner Inference: Software Interface Design Description

**Guillaume Morin-Brassard; Vincent Giroux ; DRDC Valcartier CR 2012-003 ; Recherche et développement pour la défense Canada – Valcartier ; janvier 2012.**

**Introduction ou contexte :** Afin de supporter ces activités de recherche dans le domaine du renseignement, la Section du Renseignement et Information de RDDC Valcartier développe la Plate-forme de Science et Technologie du Renseignement (ISTIP) comme un composant majeur de ses infrastructures de R&D. Lors du contrat MRI, un besoin de convertir des entités spatiales sous le format de faits situationnels a été observé car il fallait pouvoir passer ces entités spatiales à certains services requérant des faits situationnels à l'entrée. Un convertisseur d'entités spatiales vers des faits situationnels a été conçu et développé afin de supporter ce besoin. Il est exposé comme un service web SOAP sur la plateforme ISTIP.

Ce document présente la Description du Design des Interfaces Logicielles du Raisonneur Multi-Inférence et ses services, en respectant la norme IEEE 12207.

Son but est d'identifier et décrire les interfaces exposées aux autres systèmes, sous-systèmes et applications. Les interfaces identifiées et décrites devraient aussi être reflétées dans le document de description d'architecture logicielle. La description du design des interfaces logicielles est la base pour le design détaillé des processus et données des interfaces. Le document doit aussi décrire les interfaces exposées par les services du Raisonneur Multi-Inférence (incluant chaque raisonneur individuel), les flux de travail des appels aux services ainsi que les objets nécessaires à la communication inter-services.

Ce document ne décrit aucune interface interne utilisée par le MRI ou à l'intérieur des services de raisonnement avec les autres sous-composants.

**Résultats :** Cinq différents services ont été développés comprenant quatre raisonneurs individuels et un orchestrateur multi-raisonneur. Les raisonneurs qui ont été créés sont un raisonneur par cas (CBR), un raisonneur par règles (RBR), un raisonneur ontologique (DLR) et un raisonneur d'analyse cinématique et géo-spatiale (KIGAR) basé sur le module KIGAM de l'outil d'Inférence Automatisée de Faits Situationnels (ISFAR).



# Table of contents

---

Abstract .....	i
Résumé .....	i
Executive summary .....	iii
Sommaire .....	iv
Table of contents .....	v
List of figures .....	vii
List of tables .....	viii
1 General dependencies .....	1
1.1 Specific dependencies .....	1
1.1.1 KIGAR.....	1
2 External Interfaces .....	2
2.1 General interface usage .....	2
2.2 Common Interface Identification.....	3
2.2.1 Methods .....	3
2.2.2 Service Data.....	5
2.2.2.1 FactDTO and FactDefinitionDTO .....	5
2.2.2.2 ReasoningResults.....	5
2.3 KIGAR Interface Identification.....	5
2.3.1 Methods .....	6
2.3.2 Service data.....	6
2.3.2.1 FactDTO and FactDefinitionDTO .....	6
2.3.2.2 KIGARKnowHow .....	6
2.3.2.3 KIGARParameters .....	8
2.3.3 Service endpoints.....	10
2.3.4 Security .....	10
2.4 Rule-Based Reasoner (RBR) Interface Identification .....	11
2.4.1 Methods .....	11
2.4.2 Service data.....	11
2.4.2.1 FactDTO and FactDefinitionDTO .....	11
2.4.2.2 RuleBasedReasonerParameters.....	11
2.4.2.3 RuleBasedReasonerKnowHow .....	12
2.4.3 Service endpoints.....	17
2.4.4 Security .....	18
2.5 Case-Based Reasoner (CBR) Interface Identification .....	18
2.5.1 Methods .....	18
2.5.2 Service data.....	19
2.5.2.1 FactDTO and FactDefinitionDTO .....	19

2.5.2.2	CaseBasedReasonerParameters .....	19
2.5.2.3	CaseBasedReasonerKnowHow .....	20
2.5.3	Service endpoints .....	27
2.5.4	Security .....	28
2.6	Descriptive Logic Reasoner .....	28
2.6.1	Methods .....	28
2.6.2	Service data .....	29
2.6.2.1	FactDTO and FactDefinitionDTO .....	29
2.6.2.2	DescriptiveLogicReasonerParameters .....	29
	DescriptiveLogicReasonerKnowHow .....	29
2.6.3	Service endpoints .....	31
2.6.4	Security .....	32
2.7	MRI Orchestrator .....	32
2.7.1	Methods .....	32
2.7.2	Service data .....	33
2.7.2.1	FactDTO and FactDefinitionDTO .....	33
2.7.2.2	MRIOrchestratorParameters .....	33
2.7.2.3	MRIOrchestratorKnowHow .....	33
2.7.3	Service endpoints .....	35
2.7.4	Security .....	36
3	Visualization Services .....	37
3.1	Multi-Reasoners Inference Graphical Visualisation Services (MRIV) .....	37
3.1.1	Solution description .....	37
3.1.1.1	MultiReasonerInferenceGWT project .....	38
3.1.1.2	Services .....	38
3.1.1.3	Widgets and Windows .....	39
3.1.1.4	Data Modules .....	43
3.1.2	Bug report .....	43
3.1.3	Installation guide .....	43
3.1.4	Testing procedure and test results .....	44
	References .....	45
	Annex A KIGAR Analyses .....	47
	List of symbols/abbreviations/acronyms/initialisms .....	53

## List of figures

---

Figure 1: General reasoner usage workflow .....	2
Figure 2: Reasoning results class diagram .....	5
Figure 3: KIGAR KnowHow Class Diagram.....	7
Figure 4: KIGAR parameters class diagram.....	8
Figure 5: RBR parameters class diagram .....	12
Figure 6: RBR knowhow class diagram.....	13
Figure 7: Premise expression.....	14
Figure 8: Parenthesis expression .....	14
Figure 9: Negate expression .....	15
Figure 10: CBR parameters class diagram .....	19
Figure 11: CBR knowhow class diagram.....	21
Figure 12: CBR join restriction constraint .....	22
Figure 13: Template argument reference example .....	23
Figure 14: DLR parameters class diagram .....	29
Figure 15: DLR knowhow class diagram.....	30
Figure 16: MRI Orchestrator parameters class diagram.....	33
Figure 17: MRI Orchestrator knowhow class diagram.....	34
Figure 18: Multi-Reasoner Inference GWT Voiila Workspace .....	37
Figure 19: Inference Process Dashboard.....	40
Figure 20: Context Editor Window .....	41
Figure 21: Reasoning Results Window .....	42
Figure 22: Reasoning Results Window 2 .....	42
Figure 23: Multi Reasoner Inference Workspace Tool Strip.....	43

## List of tables

---

Table 1: KIGAR Service Endpoint.....	10
Table 2: KIGAR Service JNDI Names.....	10
Table 3: Rule-Based Reasoner Service Endpoint.....	17
Table 4: Rule-Based Reasoner Service JNDI Names.....	18
Table 5: Case-Based Reasoner Service Endpoint.....	27
Table 6: Case-Based Reasoner Service JNDI Names .....	27
Table 7: Descriptive Logic reasoner Service Endpoint.....	31
Table 8: Descriptive Logic Reasoner Service JNDI Names.....	31
Table 9: MRI Orchestrator Service Endpoint.....	35
Table 10: MRI Orchestrator Service JNDI Names.....	36
Table 11: KIGAR Analyses.....	47

# 1 General dependencies

---

The Multi-Reasoners Inference (MRI) services depend on the following services and therefore, to successfully deploy the MRI EAR file, the following services must be deployed alongside:

- Parameter service :
  - Description: The parameter service bean is used to fetch parameters for a given application. Every application onto the computing platform should use the preference service for updatable system wide settings.
  - Expected service mapped name :  
*Cptb/Core/Configuration/ParametersServiceBean*
  - Packaged in the *ParametersEJB.ear* file.

## 1.1 Specific dependencies

### 1.1.1 KIGAR

The Kinematic and Geospatial Analysis Reasoner (KIGAR) service can optionally use the following services if deployed alongside and available to the KIGAR service:

- Situational Facts Management (SFM) Service:
  - Description: The SFM service allows managing facts and their related atom definitions. In the MRI context, if the SFM is available, it will be used to fetch any atom definitions referenced in the input facts to be able to interpret them if not provided by the client invoker.
  - Expected service mapped name:  
*SituationalFactsManagementEJB/SituationalFactsServiceBean/remote*
  - Packages in the *SituationalFactsManagementEJB.ear* file.

All libraries required by KIGAR are packaged within the MRI EAR file.

**IMPORTANT:** Due to weak classloader isolation in JBoss 5.1, KIGAR service is conflicting with ISFAR web service. Although it is not a real issue since the MRI reasoners are intent to replace ISFAR, please make sure to undeploy ISFAR prior to deploying KIGAR.

## 2 External Interfaces

This section identifies and describes interfaces to and from external systems, subsystems, and databases. The data content for each interface, as well as the relevant physical characteristics of sending and receiving systems and referenced databases, are included. The following sections are repeated for each interface.

Each reasoner of the MRI, as well as the MRI orchestrator, are exposed as Java EJB 3 stateless session bean services locally and remotely (through RMI) for using with other Java applications. They are also exposed as standard EJB 3 SOAP Web services, allowing any system (even non-Java based ones) to easily invoke them.

### 2.1 General interface usage

The following diagram depicts, using simplified method's names described in the following sections, the user activity flow that explains how to consume any MRI reasoning service:

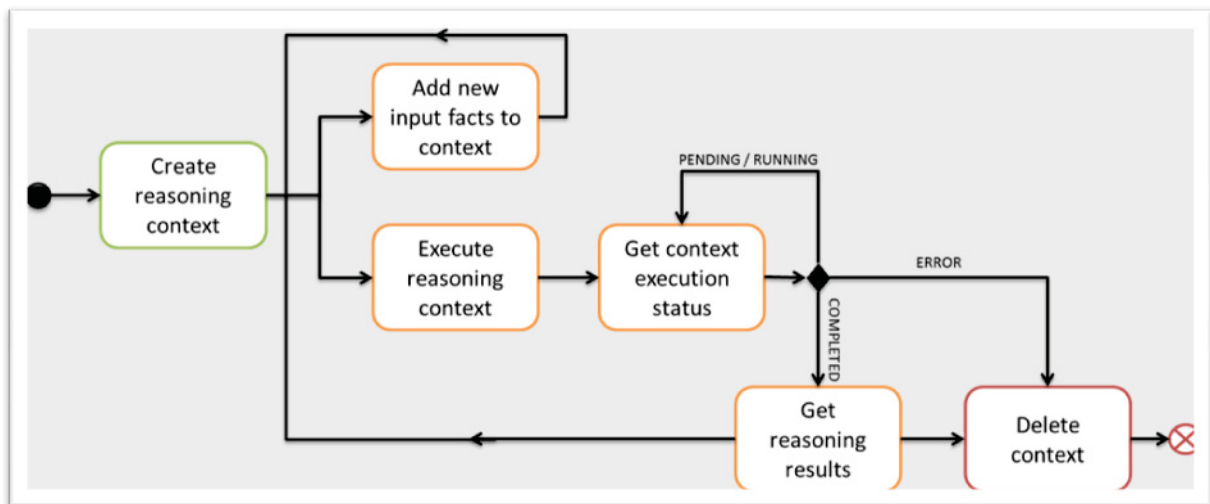


Figure 1: General reasoner usage workflow

At first, a reasoning context must be created. It basically instructs how the reasoner will handle input facts for a given domain/situation through the provided *knowhow* and parameters. It also isolates the requests from one another. The initial context can also include input facts. Once the reasoning context has been created, a context handle will be returned. This context handle will be used to perform any other action on this context.

After the context creation, the client can add new input facts at any moment. Once the context is to the satisfaction of the client, the execute method can be invoked to launch an asynchronous

reasoning process. The execution of this process is then tracked through the `get context execution status` method. This method can return one of the following statuses:

- **NEW:** This means that the reasoning context has just been instantiated and it is about to be added to the reasoning queue to be processed. This is also the default status when creating a context.
- **PENDING:** This means that the process is queue and is waiting to be processed. Therefore, the context will keep this status until the `execute` method is invoked and the reasoner dequeue the request from the processing queue;
- **RUNNING:** This means that the reasoner is currently executing the context and trying to infer new facts;
- **ERROR:** This means that an unexpected error occurred during the reasoning execution. In this case, an error message may be available in the reasoning results to get more information on what went wrong;
- **COMPLETED:** This means that the reasoning context execution has successfully completed;

Once completed, the reasoning results can be acquired through the `get reasoning results` method. This method will return inferred facts and any feedback messages to inform the client about what happened during the reasoning (conversion warnings, general information, non-fatal errors, etc.). Afterward, the client can decide to add new input facts to the context and execute another inference loop.

Finally, when the reasoning is obsolete, the `delete context` method allows deleting the context and cleaning any resources associated with it.

Please refer to the *Multi-Reasoners Inference Software Architecture Description* for more information on the internal workflow and logic of each action described above.

## 2.2 Common Interface Identification

All MRI services adopt the same base interface. This interface allows the caller to send a query and retrieve tasks progression and results. Due to the nature of the service, the caller cannot wait for the process to complete in a single method call, as the web service call would timeout and the information would be lost.

To this end, the service's interface provides methods to both launch reasoning and consume results. Upon new reasoning context creation, the service returns a context handle (identifier) to the caller, which will be needed when launching a reasoning execution or for any other operations on this context (`get current processing status`, `add input facts`, `fetch results`, etc.).

### 2.2.1 Methods

This section describes the interface methods that are common to every reasoner of the MRI, as well as the MRI orchestrator. Indeed, only the `createContext` method differs from one reasoner to another due to different *knowhow* and parameters.

- void addFacts(UUID<sup>1</sup> contextHandle, Collection<FactDTO> facts, Collection<FactDefinitionDTO> factDefinitions)

This method allows adding input facts to an existing context at any moment. However, like for the `createContext` method, this method doesn't launch the reasoning execution. The client must explicitly invoke the `executeAsync` method afterward to do so.

This method may throw an *InvalidHandleException* if the specified context handle does not exist or if a *ReasonerException* fatal error occurs.

- void executeAsync(UUID<sup>1</sup> contextHandle, String jndiToReasonerICallback)<sup>2</sup>  
This method allows adding a reasoning context to the processing queue to be executed by the reasoner afterward. The *jndiToReasonerICallback* parameter allows specifying a JNDI string of a service implementing the *ICallback* interface. If set, this service will be invoked after the execution of the context. However, please note that the service JNDI must be available by the reasoner (either be located within the same JBoss instance or properly registered in the JBoss instance otherwise). If the *jndi* callback parameter is not provided, the client must follow the execution and retrieve the results asynchronously using the *getStatus* and *getReasoningResults* methods explained below since the context will not be executed within the scope of this method invocation.
- void execute(String contextHandle)<sup>3</sup>  
This method allows adding a reasoning context to the processing queue to be executed by the reasoner afterward. The client must then follow the execution and retrieve the results asynchronously using the *getStatus* and *getReasoningResults* methods explained below since the context will not be executed within the scope of this method invocation.
- ReasonerContextStatus getStatus(UUID<sup>1</sup> contextHandle)  
This method allows querying the current execution status of a reasoning context based on the provided context handle. The context status can be one of the following values:
  - NEW: This is the default status when the context is created or facts are added to the context. This means that the context has never been executed (the `executeAsync` or `execute` method has never been invoked yet).
  - PENDING: This means that the `executeAsync` (or `execute`) method has been invoked and the context is currently in the processing queue waiting to be executed.

---

<sup>1</sup> Please note that in the case of the Web Service interface, UUID objects are replaced by Strings, which can be easily transferred using SOAP standard.

<sup>2</sup> This method is only available through the Java local or remote interface (through RMI). Please refer to the *execute* method for the Web Service interface equivalent method.

<sup>3</sup> This method is only available through the Web Service interface. Please refer to the *executeAsync* method for the remote or local Java interface.



- **RUNNING:** The reasoner is currently executing this reasoning context.
  - **COMPLETED:** The reasoner has successfully executed the reasoning context and new inferred facts may be available.
  - **ERROR:** An error occurred during the context execution. The *getReasoningResults* method can be invoked to retrieve error messages if any.
- **ReasoningResults getReasoningResults(UUID1 contextHandle)**  
This method allows fetching inferred facts as well as any messages created by the reasoner. The messages can either be informative, warnings about some information missing for instance, or even error messages.
  - **void deleteContext(UUID1 contextHandle)**  
Since the reasoning contexts are managed asynchronously, they are persisted and retrieved from an invocation to another. Therefore, the reasoning contexts are always available and occupying physical hard drive space on the server. This method allows to permanently delete a reasoning context once it is deprecated.

## 2.2.2 Service Data

### 2.2.2.1 FactDTO and FactDefinitionDTO

These objects are imported directly from the SFM service API containing the fact model. Please refer to the section 3.7 of the document JCDS-CTB-TA69-310-0426-DR-v0.7.doc for more information about these objects.

### 2.2.2.2 ReasoningResults

The following diagram depicts the class structure of the ReasoningResults class:

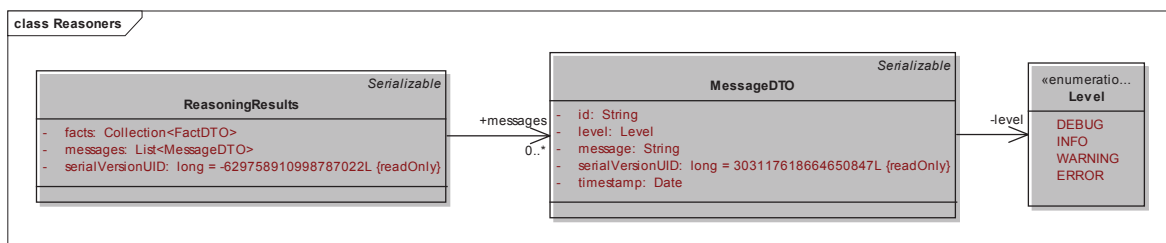


Figure 2: Reasoning results class diagram

The ReasoningResults contains basically a list of inferred facts and a list of messages from the reasoner.

Each MessageDTO contains a timestamp, a message level and the text message.

## 2.3 KIGAR Interface Identification

The Kinematic and Geospatial Analysis Reasoner (KIGAR) service interface definition is an input and output interface which allows the caller to send a query and retrieve tasks progression

and results. Due to the nature of the service, the caller cannot wait for the process to complete in a single method call, as the web service call would timeout and the information would be lost.

To this end, the service's interface provides methods to both launch reasoning and consume results. Upon new reasoning context creation, the service returns a context handle (identifier) to the caller, which will be needed when launching a reasoning execution or for any other operations on this context (get current processing status, add input facts, fetch results, etc.).

### 2.3.1 Methods

- UUID<sup>4</sup> createContext(Collection<FactDTO> facts, Collection<FactDefinitionDTO> factDefinitions, KIGARKnowHow knowHow, KIGARParameters parameters)

This method allows creating a reasoning context. It is the first step for performing automatic reasoning. It initializes a specific "working context" with the provided facts, atom definitions, knowhow and parameters, and then return the initialized context identifier. This identifier will then be used to perform any other actions on this context.

Please note that this method doesn't launch the reasoning execution. The client must explicitly invoke the *executeAsync* method (or *execute* method for the WS) afterward to do so.

This method may throw a *ReasonerException* if anything goes wrong during the context creation.

### 2.3.2 Service data

#### 2.3.2.1 FactDTO and FactDefinitionDTO

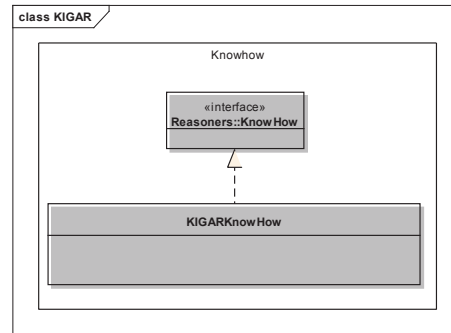
These objects are imported directly from the SFM service API containing the fact model. Please refer to the section 3.7 of the document JCDS-CTB-TA69-310-0426-DR-v0.7.doc for more information about these objects.

#### 2.3.2.2 KIGARKnowHow

The following diagram depicts the class structure of the KIGARKnowHow class:

---

<sup>4</sup> Please note that in the case of the Web Service interface, UUID objects are replaced by Strings, which can be easily transferred using SOAP standard.



*Figure 3: KIGAR KnowHow Class Diagram*

KIGAR doesn't require any specific knowhow since the knowledge is implemented directly in the different geospatial analyses. However, in order to fulfill the common reasoning interface, an empty class has been implemented. This also leaves the opportunity to add any knowhow to the reasoner in future improvements of the service.

### 2.3.2.3 KIGARParameters

The following diagram depicts the class structure of the KIGARParameters class:

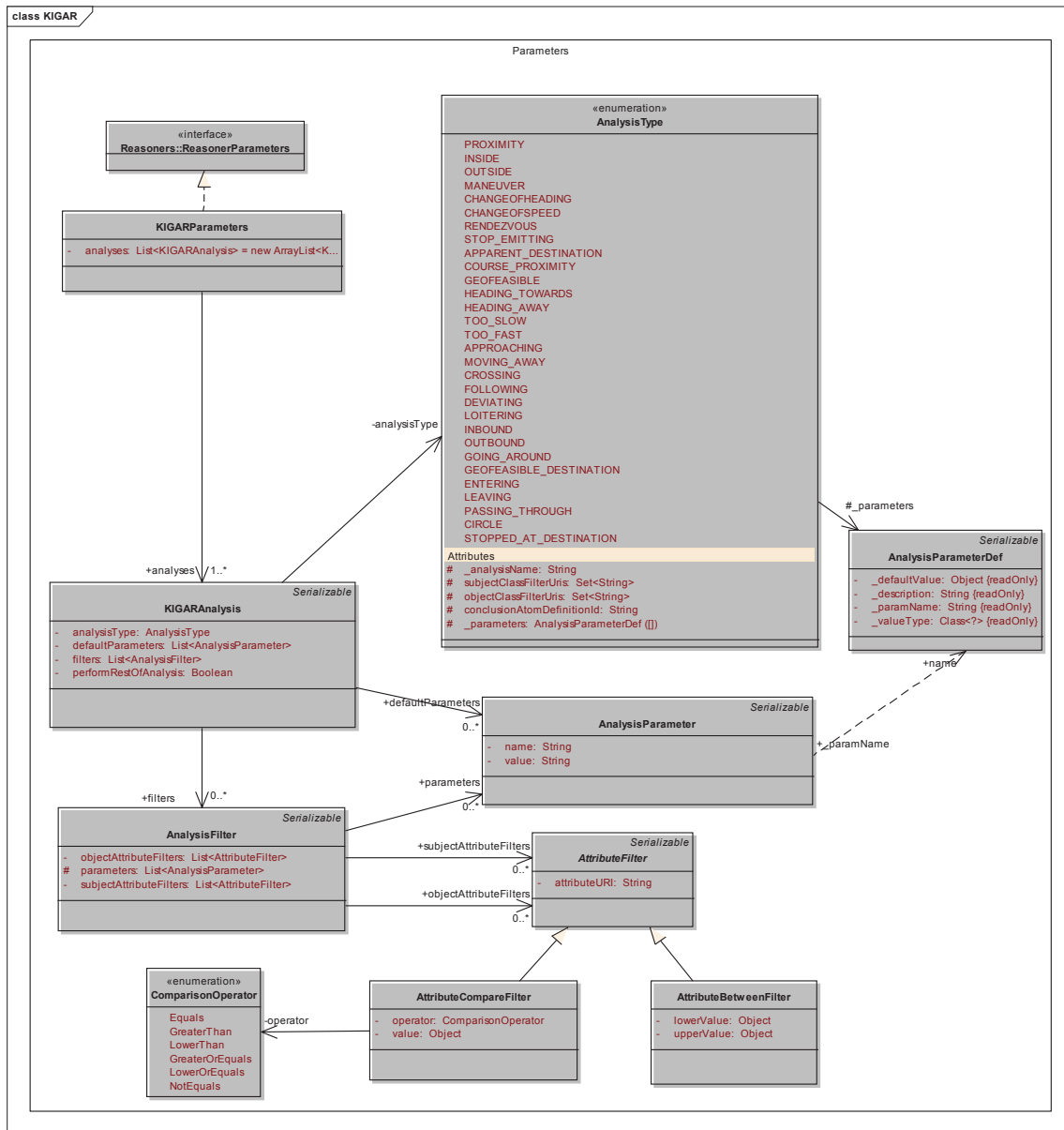


Figure 4: KIGAR parameters class diagram

Here is a brief description of classes presented this diagram:

- **KIGARParameters:** This class only contains a list of KIGARAnalysis objects stating which analyses to perform;

- **KIGARAnalysis:** This class defines a specific analysis to execute and optionally allows performing the analysis on specific subjects/objects based on a set of filters.
  - The *analysisType* field defines the analysis to execute. The value must be one of the AnalysisType enumeration value. Please refer to the *Multi-Reasoners Inference Software Architecture Description* for the full list of analyses.
  - The *defaultParameters* field allows overriding default analysis parameters. They are simply defined by a list of key-value pairs. Please also refer to the *Multi-Reasoners Inference Software Architecture Description* about which parameter is available per analysis.
  - The *filters* field allows defining groups of subjects/objects that will be evaluated with different parameters. They are selected in a similar way than a search query – by defining a set of attributes that must be matched against subjects/objects and a set of parameter values to apply to them.
  - The *performRestOfAnalysis* flag allows indicating whether subjects/objects that have not been matched by filters should be evaluated or not. If set to true, they will be evaluated with the overridden *defaultParameters* or the analysis default values.  
Please note that analyses can be simply configured by providing no filters and setting *performRestOfAnalysis* flag to true. In this case, every subjects/objects found in the input facts will be evaluated with the same parameter values.
- **AnalysisFilter:** This class defines a set of subjects/objects attributes restrictions used to select a subset of the subjects/objects found in the input facts and a set of parameter values to apply to them when performing the analysis.
  - The *subjectAttributeFilters* field is a list of attributes criteria that is applied to subjects found in the input facts. Each filter is a conjunction with other filters (filter1 AND filter2 AND filter3).
  - The *objectAttributeFilters* field is a list of attributes criteria that is applied to objects found in the input facts. Each filter is a conjunction with other filters (filter1 AND filter2 AND filter3).
  - The *parameters* field allow overriding parameter values. The precedence order is the following: when searching for a parameter value, the parameter key will be searched for in the *parameters*. If not found, the parameter will then be searched in the *defaultParameters* field of the parent KIGARAnalysis object. If still not found, the analysis will use a default system value.  
Please refer to the MRI-Software Architecture Description for the complete list of parameters available for each analysis.
- **AttributeCompareFilter:** This kind of filter must define an attribute URI, an attribute value and an operator used to compare the subject/object attribute value with the defined value.
- **AttributeBetweenFilter:** This kind of filter must define an attribute URI, an attribute value lower and upper bound. The restriction is inclusive (value must be

greater than or equals to the lower bound and lower than or equals to the upper bound).

### 2.3.3 Service endpoints

The Web Service interface endpoint has the following scheme:

`http://host:port/istip/mri-kigar/KIGARWS`

where the *port* is usually 8080 in the case of JBoss

Moreover, the WSDL is available by simply adding “?wsdl” at the end of the previous address scheme when deployed in JBoss.

The Web Service interface endpoint has also been registered to the ISTIP UDDI:

*Table 1: KIGAR Service Endpoint*

Service Name	Value
KIGAR Service	<code>http://10.9.1.200:8080/istip/mri-kigar/KIGARWS</code>

KIGAR service is also available through Java remote and local interfaces through JBoss using the following JNDI names:

*Table 2: KIGAR Service JNDI Names*

JNDI Name	Interface	Type
MRI/KIGARReasonerBean/local	<code>ca.gc.rddc.kigar.KIGARReasonerLocal</code>	Local
MRI/KIGARWSBean/remote- <code>ca.gc.rddc.kigar.KIGARReasonerRemote</code>	<code>ca.gc.rddc.kigar.KIGARReasonerRemote</code>	Remote

### 2.3.4 Security

KIGAR service can be secured through OpenSSO authentication if deployed with ant using the following parameter value:

`apply-security = true`

Actually, the service is only configured to allow access to any authenticated user when deployed with the above parameter value.

Please refer to the document *Contract Number: W7701-5-3182, Task Authorization 69, SOFTWARE INSTALLATION GUIDE (SIG) – SSO* for more information on how to configure OpenSSO with JBoss.

Please also refer to the document *User Single Sign-on – Development Report – Group C Deliverable 1 – JCDS-CTB-TA69-310-0432-DR* for more information about the service security configuration.

## 2.4 Rule-Based Reasoner (RBR) Interface Identification

### 2.4.1 Methods

- UUID<sup>5</sup> createContext(Collection<FactDTO> facts, Collection<FactDefinitionDTO> factDefinitions, RuleBasedReasonerKnowHow knowHow, RuleBasedReasonerParameters parameters)

This method allows creating a reasoning context. It is the first step for performing automatic reasoning. It initializes a specific "working context" with the provided facts, atom definitions, knowhow and parameters, and then return the initialized context identifier. This identifier will then be used to perform any other actions on this context.

Please note that this method doesn't launch the reasoning execution. The client must explicitly invoke the *executeAsync* method (or *execute* method for the WS) afterward to do so.

This method may throw a *ReasonerException* if anything goes wrong during the context creation (for example if there is an invalid rule in the knowhow).

### 2.4.2 Service data

#### 2.4.2.1 FactDTO and FactDefinitionDTO

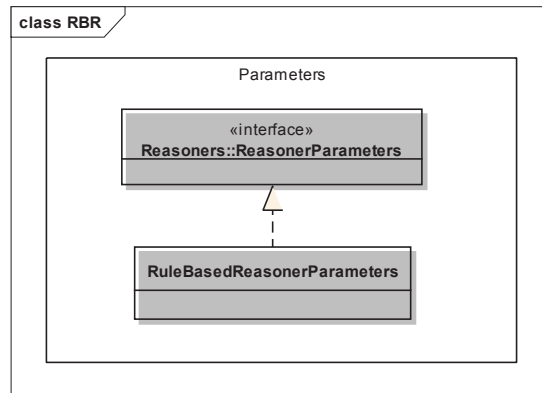
These objects are imported directly from the SFM service API containing the fact model. Please refer to the section 3.7 of the document JCDS-CTB-TA69-310-0426-DR-v0.7.doc for more information about these objects.

#### 2.4.2.2 RuleBasedReasonerParameters

The following diagram depicts the class structure of the RuleBasedReasonerParameters class:

---

<sup>5</sup> Please note that in the case of the Web Service interface, UUID objects are replaced by Strings, which can be easily transferred using SOAP standard.



*Figure 5: RBR parameters class diagram*

RBR doesn't require any specific execution parameter. However, in order to fulfill the common reasoning interface, an empty class has been implemented. This also leaves the opportunity to add any parameters to the reasoner in future improvements of the service.

#### **2.4.2.3 RuleBasedReasonerKnowHow**

The following diagram depicts the class structure of the RuleBasedReasonerKnowHow class:



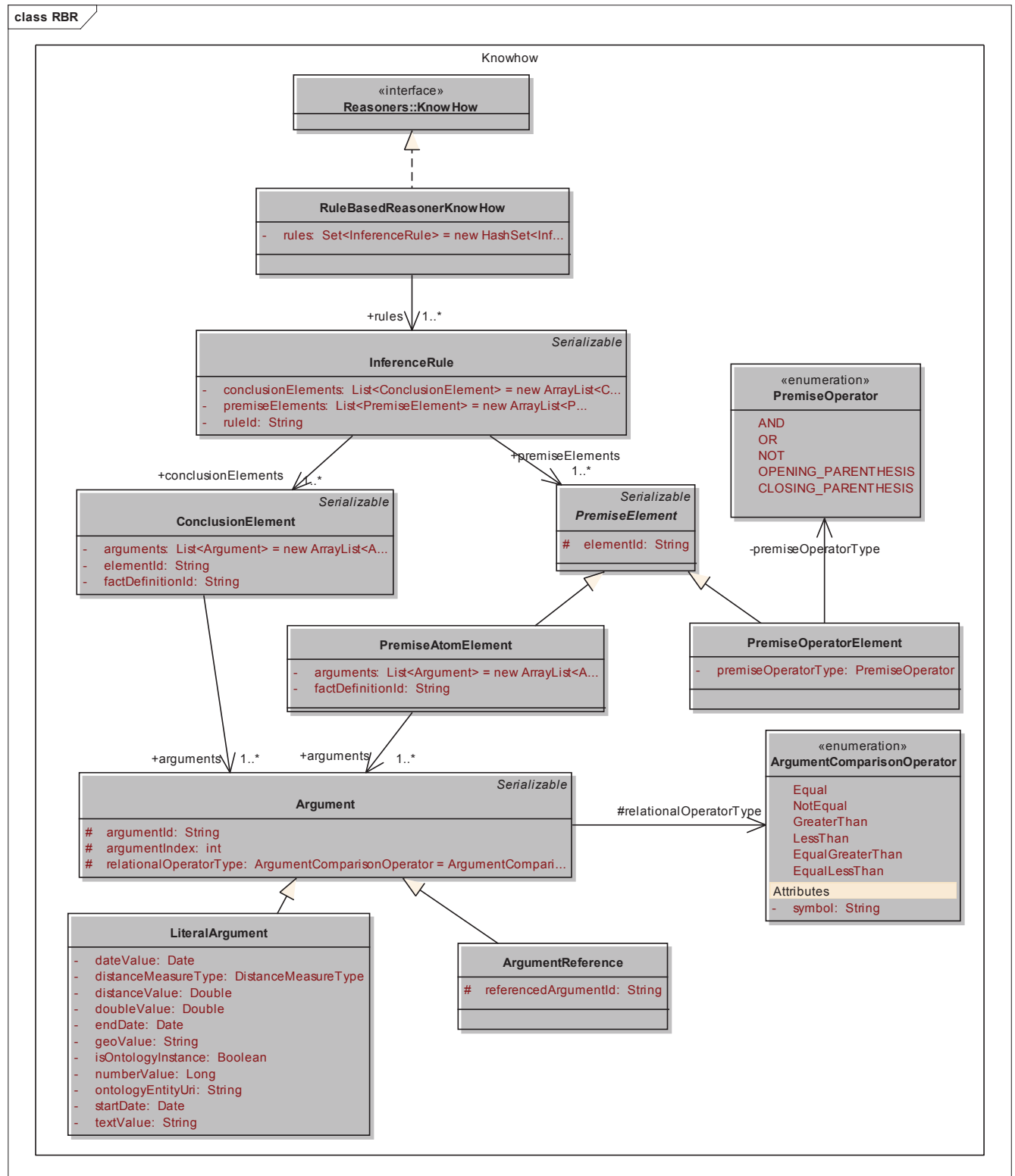


Figure 6: RBR knowhow class diagram

Here is a brief description of classes presented this diagram:

- **RuleBasedReasonerKnowHow:** This class simply contains a list of inference rules
- **InferenceRule:** This class defines an inference rule by using three fields:
  - The *ruleId* field is used to identify rules within the same context. Therefore, the *ruleId* should be unique among the inference rules of the same context.
  - The *premiseElements* field contains the rule premise expression composed of atom (fact) restrictions and operators. When this expression is matched among the facts in the context, the conclusion is generated.
  - The *conclusionElements* field basically define which facts will be created when the premise expression is matched among facts in the context and how their arguments are valued (either predefined values or values based on the facts that have been matched with the rule premises).
- **PremiseElement:** Abstract class. A premise element can either be a *PremiseAtomElement* – an atom restriction, or a *PremiseOperatorElement* – an operator to build complex expressions of AND, OR, NOT, etc. with atom restrictions. However, please note that the premise elements sequence is very important and must respect the following syntax:

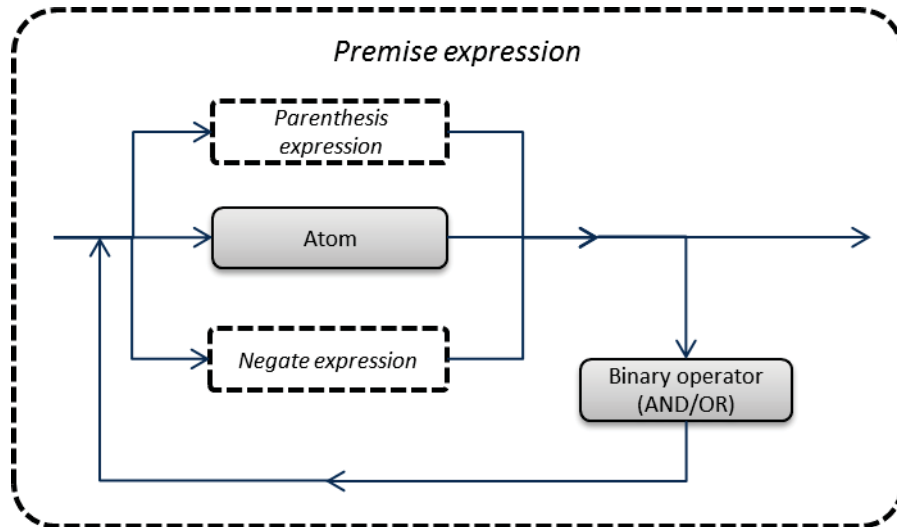


Figure 7: Premise expression

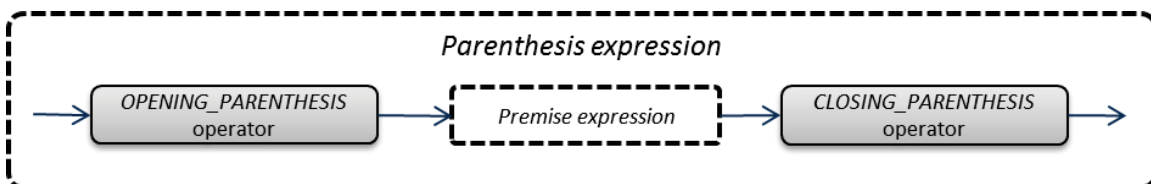


Figure 8: Parenthesis expression

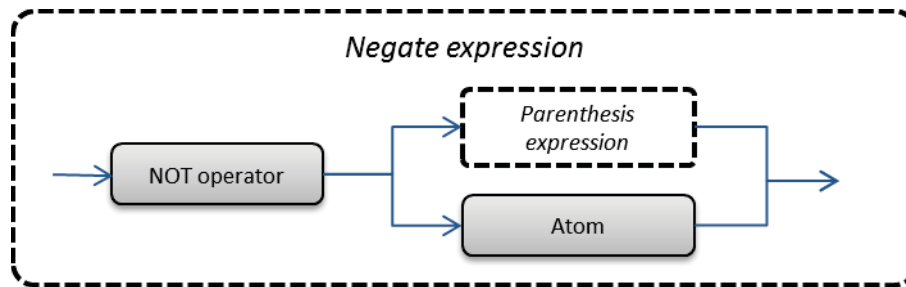


Figure 9: Negate expression

- **PremiseAtomElement:** This class defines restriction on a fact that must be found within the context facts.
  - The *factDefinitionId* field first identify which “kind” of fact that must be matched based on its fact definition identifier. This property is mandatory.
  - The *arguments* field defines a set of restrictions on the fact arguments
- **Argument:** This base class can only be used in the rule premises. It allows defining a fact argument in the premises without defining any restriction on it. It is meant to be referenced by another *ArgumentReference* since the “binding” is performed using the *argumentId* field.
  - The *argumentId* field must uniquely identify the argument restriction among all arguments constituting the inference rule. It is mainly used to be referred to by other argument restrictions in the premises or conclusions (through *ArgumentReference*)
  - The *argumentIndex* field is used to identify the argument within the fact arguments (mainly since arguments without restrictions can be omitted in the premises);
- **LiteralArgument:** This class defines a restriction on a fact argument value when used in the rule premises or as a predetermined argument value when used in the rule conclusion.
  - The *argumentId* field must uniquely identify the argument restriction among all arguments constituting the inference rule. It is mainly used to be referred to by other argument restrictions in the premises or conclusions (through *ArgumentReference*)
  - The *argumentIndex* field is used to identify the argument within the fact arguments (mainly since arguments without restrictions can be omitted in the premises);
  - The *relationalOperatorType* field allows selecting the comparison method when used in the premises. If the argument is in the conclusion section, this field is simply ignored.
    - *Equal*: the fact argument value must match exactly the value restriction set in the premise literal argument

- *NotEqual*: the fact argument value must not match the value restriction set in the premise literal argument
- *GreaterThan*: the fact argument value must be exclusively greater than the value restriction set in the premise literal argument
- *LessThan*: the fact argument value must be exclusively less than the value restriction set in the premise literal argument
- *EqualGreaterThan*: the fact argument value must be either exactly equal to or greater than the value restriction set in the premise literal argument
- *EqualLessThan*: the fact argument value must be either exactly equal to or less than the value restriction set in the premise literal argument
- The other fields (*dateValue*, *doubleValue*, etc.) are used to specify the argument value based on the argument value type. Please refer to the fact argument for more information about these fields.
- **ArgumentReference**: This class defines either a restriction on a fact argument value by comparing it with another argument value when used in the rule premises, or an argument value that will be copied from a fact matched in the premises when used in the rule conclusion.
  - The *argumentId* field must uniquely identify the argument restriction among all arguments constituting the inference rule. It is mainly used to be referred to by other argument restrictions in the premises or conclusions (through *ArgumentReference*)
  - The *argumentIndex* field is used to identify the argument within the fact arguments (mainly since arguments without restrictions can be omitted in the premises);
  - The *relationalOperatorType* field allows selecting the comparison method when used in the premises. If the argument is in the conclusion section, this field is simply ignored.
    - *Equal*: the fact argument value must match exactly the value found in the referenced argument
    - *NotEqual*: the fact argument value must not match the value found in the referenced argument
    - *GreaterThan*: the fact argument value must be exclusively greater than the value found in the referenced argument
    - *LessThan*: the fact argument value must be exclusively less than the value found in the referenced argument
    - *EqualGreaterThan*: the fact argument value must be either exactly equal to or greater than the value found in the referenced argument
    - *EqualLessThan*: the fact argument value must be either exactly equal to or less than the value found in the referenced argument

- The *referencedArgumentId* field is used to determine which argument is referenced by this restriction by providing its corresponding *argumentId* identifier. When used in the rule premises, the value found in the referenced argument will be used for the comparison restriction. When used in the rule conclusion, it must refer to an argument defined in the rule premises and will copy the value matched at runtime by this premise restriction in the conclusion fact argument.
- **PremiseOperatorElement:** This class allows defining complex premises conditions by adding operators between premise atom restrictions.
  - The *premiseOperatorType* field is used to select the operator to use:
    - *AND*: Both expressions surrounding the operator must be matched
    - *OR*: Only one of the two expressions surrounding the operator needs to be matched
    - *NOT*: The expression immediately following the operator must not be matched
    - *OPENING\_PARENTHESIS*: opening parenthesis “(“ used to explicit operator precedence
    - *CLOSING\_PARENTHESIS*: closing parenthesis “)” used to close a group of expressions starting by an opening parenthesis
- **ConclusionElement:** This class allows defining which facts will be created as inferred facts if the rule premise expression is matched against the context facts, and how their arguments will be valued.
  - The *factDefinitionId* field first identify which “kind” of fact that will be generated as inferred facts using its fact definition identifier.
  - The *arguments* field is used to determine how each argument of the inferred facts will be valued using *LiteralArgument* objects for predefined values or *ArgumentReference* objects for values based on arguments matching the premise expression.

### 2.4.3 Service endpoints

The Web Service interface endpoint has the following scheme:

`http://host:port/istip/mri-rbr/RuleBasedReasonerWS`

where the *port* is usually 8080 in the case of JBoss

Moreover, the WSDL is available by simply adding “?wsdl” at the end of the previous address scheme when deployed in JBoss.

The Web Service interface endpoint has also been registered to the ISTIP UDDI:

*Table 3: Rule-Based Reasoner Service Endpoint*

Service Name	Value
Rule Based Reasoner Service	<code>http://10.9.1.200:8080/istip/mri-rbr/RuleBasedReasonerWS</code>

The RBR service is also available through Java remote and local interfaces through JBoss using the following JNDI names:

*Table 4: Rule-Based Reasoner Service JNDI Names*

JNDI Name	Interface	Type
MRI/RuleBasedReasonerBean/local	ca.gc.rddc.rulebasedreasoner.RuleBasedReasonerLocal	Local
MRI/RuleBasedReasonerBean/remote- ca.gc.rddc.rulebasedreasoner.RuleBasedReasonerRemote	ca.gc.rddc.rulebasedreasoner.RuleBasedReasonerRemote	Remote

#### 2.4.4 Security

The RBR service can be secured through OpenSSO authentication if deployed with ant using the following parameter value:

apply-security = true

Actually, the service is only configured to allow access to any authenticated user when deployed with the above parameter value.

Please refer to the document *Contract Number: W7701-5-3182, Task Authorization 69, SOFTWARE INSTALLATION GUIDE (SIG) – SSO* for more information on how to configure OpenSSO with JBoss.

Please also refer to the document *User Single Sign-on – Development Report – Group C Deliverable 1 – JCDS-CTB-TA69-310-0432-DR* for more information about the service security configuration.

## 2.5 Case-Based Reasoner (CBR) Interface Identification

### 2.5.1 Methods

- UUID<sup>6</sup> createContext(Collection<FactDTO> facts, Collection<FactDefinitionDTO> atomDefinitions, CaseBasedReasonerKnowHow knowHow, CaseBasedReasonerParameters parameters)

<sup>6</sup> Please note that in the case of the Web Service interface, UUID objects are replaced by Strings, which can be easily transferred using SOAP standard.

This method allows creating a reasoning context. It is the first step for performing automatic reasoning. It initializes a specific "working context" with the provided facts, atom definitions, knowhow and parameters, and then return the initialized context identifier. This identifier will then be used to perform any other actions on this context.

Please note that this method doesn't launch the reasoning execution. The client must explicitly invoke the *executeAsync* method (or *execute* method for the WS) afterward to do so.

This method may throw a *ReasonerException* if anything goes wrong during the context creation (for example if there is an invalid template in the knowhow).

## 2.5.2 Service data

### 2.5.2.1 FactDTO and FactDefinitionDTO

These objects are imported directly from the SFM service API containing the fact model. Please refer to the section 3.7 of the document JCDS-CTB-TA69-310-0426-DR-v0.7.doc for more information about these objects.

### 2.5.2.2 CaseBasedReasonerParameters

The following diagram depicts the class structure of the CaseBasedReasonerParameters class:

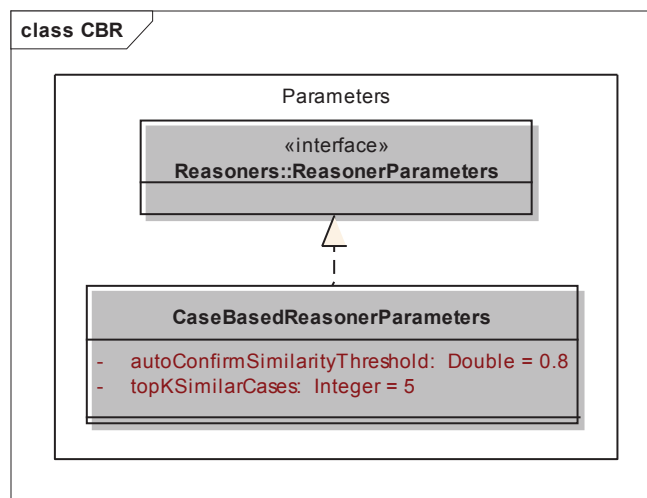


Figure 10: CBR parameters class diagram

Here is a brief description of the classes presented in this diagram:

- **CaseBasedReasonerParameters:** This class defines the general parameters for the case-based reasoner. It only contains two (2) parameters:
  - The *topKSimilarCases* field defines the maximal number of cases to keep for a given situation (subset of input facts) matching a given situation

template. If set to null, all similar cases will be assigned to the inferred facts;

- The *autoConfirmSimilarityThreshold* field defines the minimal similarity value required to automatically consider a given situation (subset of input facts) as similar to a given case, and therefore generate the appropriate inferred facts based on the situation template. The value should be between 0 and 1 (as case similarity factors are normalized within this value range).

Both parameters are applied to the cases comparison set. Therefore, if both parameters are set, only the top K cases matching the given situation description with a similarity factor higher (or equal) than the threshold will be retained.

### **2.5.2.3 CaseBasedReasonerKnowHow**

The following diagram depicts the class structure of the CaseBasedReasonerKnowHow class:



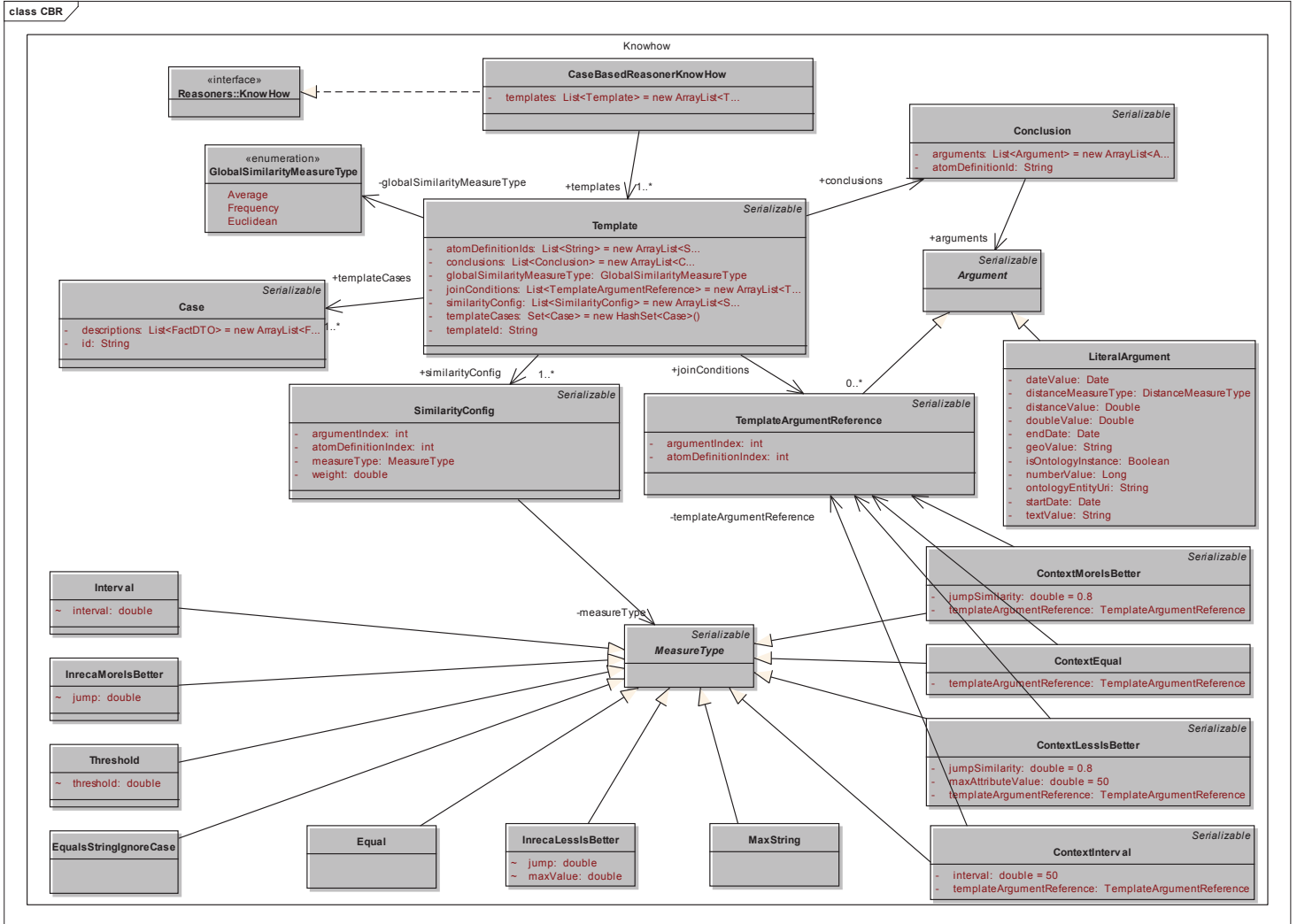


Figure 11: CBR knowhow class diagram

Here is a brief description of the classes presented in this diagram:

- **CaseBasedReasonerKnowHow:** This class basically contains a list of situation templates.
- **Template:** This class defines how facts are organized into situations and then evaluated against other situations.
  - The *templateId* field uniquely identify the template;
  - The *atomDefinitionIds* field defines the list of facts constituting the situation. The list is ordered to ease referencing arguments by using the fact index. When building situations out of input facts, the converter will try to match the list of fact definitions in the template to build a situation (with join conditions if any);
  - The *joinConditions* field defines a list of *TemplateArgumentReference* to enforce situation facts grouping by argument having the same value.

Here is an example. Based on the list of *atomDefinitionIds*, situations with any combination of atom definition 1, 2 and 3 could be made. However, the *joinConditions* adds to these facts a join restriction.

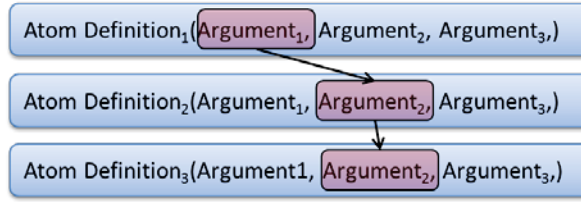


Figure 12: CBR join restriction constraint

In the example above, the *joinConditions* field would contain the following values: { [fact index: 1, argument index: 1], [fact index 2, argument index: 2], [fact index: 3, argument index: 2] }. This will make sure that the argument values of the first fact and first argument will match the second argument of the second fact, as well as the second argument of the third fact. Typical cases of join conditions are to ensure that subjects of each fact are the same.

- The *similarityConfig* field defines local similarity measures – which fact arguments should be used for situation comparison, and which comparison algorithm to use.
- The *globalSimilarityMeasureType* field selects how each individual similarity measure will be computed to form a unified global similarity measure. There are currently three (3) algorithms implemented:
  - *Average*: compute the average local similarity of all compared fields:

$$\text{Global similarity} = \frac{\sum_{k=1}^n \text{weight}_k \times \text{localSimilarity}_k}{n}$$

- *Euclidean*: compute the Euclidean similarity of all compared fields:

$$\text{Global similarity} = \sqrt{\frac{\sum_{k=1}^n \text{weight}_k \times \text{localSimilarity}_k^2}{\sum_{k=1}^n \text{weight}_k}}$$

- *Frequency*: compute the average local similarity by normalizing them to 0 or 1 if the similarity is greater than 0:

$$\text{Global similarity} = \frac{\sum_{k=1}^n x}{n} \text{ where } x = \begin{cases} 0, & \text{if } \text{localSimilarity}_k \leq 0 \\ 1, & \text{if } \text{localSimilarity}_k > 0 \end{cases}$$

- The *templateCases* field defines one or more case(s) constituting the case base for this template. Please note that these cases must comply with the structure defined in the template (atom definition ids, etc.). If no case is

provided, the template will not be used since the reasoner will have no case to compare the situation descriptions with.

- The *conclusions* field defines one or more fact(s) that will be generated when a description matches a case and is respecting the *autoConfirmSimilarityThreshold* defined previously. The fact arguments can be hard coded or can be defined as *TemplateArgumentReference* which will copy the value of an argument of the description into the conclusion fact.
- **TemplateArgumentReference:** This class allows referencing arguments by their atom definition (or corresponding fact) index within the template and the argument index within this fact. The following figure presents how this reference is achieved:

		Argument index		
		1	2	3
Atom definition index	1	Atom Definition <sub>1</sub> (Argument <sub>1</sub> , Argument <sub>2</sub> , Argument <sub>3</sub> )		
	2	Atom Definition <sub>2</sub> (Argument <sub>1</sub> , Argument <sub>2</sub> )		
	3	Atom Definition <sub>1</sub> (Argument <sub>1</sub> , Argument <sub>2</sub> , Argument <sub>3</sub> )		

Atom definition index: 3  
Argument index: 1

Figure 13: Template argument reference example

- The *atomDefinitionIndex* field select the atom definition (or fact) within the situation template by its position in the list (starting at 1). Here, the position is used since the same atom definition may be used twice in the template and each may not have the same meaning in the situation.
- The *argumentIndex* field select the argument within the atom definition (or fact) by its position (starting at 1).
- **SimilarityConfig:** This class defines a local similarity configuration for comparing situation descriptions against the template cases. The local similarity will be evaluated between two corresponding<sup>7</sup> arguments.
  - The *atomDefinitionIndex* field select the atom definition (or fact) within the situation template by its position in the list (starting at 1) exactly like for the *TemplateArgumentReference* class.
  - The *argumentIndex* field select the argument within the atom definition (or fact) by its position (starting at 1) exactly.

<sup>7</sup> The argument of the case is compared with the argument from the situation description having the same fact index and argument index.

- The *measureType* field defines how the local similarity will be computed.
  - The *weight* field allows defining different weight to local similarities (when using a global similarity measure using it). For example, if a the weight of a local similarity is set to “2” and all other local similarities weight is set to “1”, the first local similarity will have twice the weight of each other similarity in the global similarity measure.
- **Conclusion:** This class defines facts that will be generated if a description is similar to a case.
  - The *atomDefinitionId* field defines the type of atom definition that will be generated.
  - The *arguments* field defines the arguments of the fact that will be generated which can be hard-coded (*LiteralArguments*) or can be copied from the description with *TemplateArgumentReference*.
- **Argument:** This class defines an argument which can be used in a template conclusion. There are two types of arguments: hard-coded (*LiteralArguments*) or copied from the description (*TemplateArgumentReference*).
- **Literal Argument:** This class defines the content of a hard-coded argument in a template conclusion.
  - The *dateValue* field defines the date value of an argument.
  - The *distanceMeasureType* field defines the distance measure type value of an argument.
  - The *distanceValue* field defines the distance value of an argument.
  - The *doubleValue* field defines the double value of an argument.
  - The *endDate* field defines the end date value of a date range argument.
  - The *geoValue* field defines the geometry value as WKT of an argument.
  - The *isOntologyInstance* field defines if the ontologyEntityUri field of the argument represents an individual or a class.
  - The *numberValue* field defines the long value of an argument.
  - The *ontologyEntityUri* field defines the ontology entity uri value of an argument.
  - The *startDate* field defines the start date value of a date range argument.
  - The *textValue* field defines the text value of an argument.
- **Equal:** This local similarity function class compares two argument values and return 1 if they are equals and 0 otherwise;
- **EqualsStringIgnoreCase:** This local similarity function class compares two argument values and return 1 if they are equals (without comparing case) and 0 otherwise. However, the argument value must be of type String to use this similarity function.
- **InrecaLessIsBetter:** This local similarity function class compares two argument values – which must be numerical – and return the following value:

$$\text{localSimilarity} = \begin{cases} 1, & \text{if } \text{caseValue} \leq \text{queryValue} \\ 0, & \text{if } \text{caseValue} \geq \text{maxValue} \\ \text{jump} \times \frac{\text{maxValue} - \text{caseValue}}{\text{maxValue} - \text{queryValue}}, & \text{otherwise} \end{cases}$$

where *caseValue* is the argument value of the evaluated case, *queryValue* is the argument value of the evaluated situation description, and *jump* and *maxValue* are the function parameters.

- The *maxValue* field defines the reference maximum value
- The *jump* field defines the factor to use if the situation description value (*queryValue*) is less than the case value and the *maxValue*. This value should be between 0 and 1 inclusively.
- **InrecaMoreIsBetter:** This local similarity function class compares two argument values – which must be numerical – and return the following value:

$$\text{localSimilarity} = \begin{cases} 1, & \text{if } \text{caseValue} \geq \text{queryValue} \\ \text{jump} \times \left(1 - \frac{\text{queryValue} - \text{caseValue}}{\text{queryValue}}\right), & \text{otherwise} \end{cases}$$

where *caseValue* is the argument value of the evaluated case, *queryValue* is the argument value of the evaluated situation description, and *jump* is the function parameter.

- The *jump* field defines the factor to use if the situation description value (*queryValue*) is greater than the case value. This case should be between 0 and 1 inclusively.
- **Interval:** This local similarity function class compares two argument values – which must be numerical – and return the following value:

$$\text{localSimilarity} = 1 - \left(\frac{|\text{caseValue} - \text{queryValue}|}{\text{interval}}\right)$$

where *caseValue* is the argument value of the evaluated case, *queryValue* is the argument value of the evaluated situation description, and *interval* is the function parameter.

- The *interval* field should define the maximal possible difference between both values.
- **MaxString:** This local similarity function class compares two arguments – which should be of text type – and return a value depending on the biggest substring that belong to both strings.

$$\text{localSimilarity} = \begin{cases} 1, & \text{if } \text{caseValue} = \text{queryValue} \\ \frac{\text{maxSubStringLength}}{\text{maxLength}}, & \text{otherwise} \end{cases}$$

where *caseValue* is the argument value of the evaluated case, *queryValue* is the argument value of the evaluated situation description, *maxLength* is the maximal string length of both strings and *maxSubStringLength* is the length of the biggest substring that belong to both strings.

- **Threshold:** This local similarity function class compares two arguments – which should be numerical – and return the following value:

$$\text{localSimilarity} = \begin{cases} 0, & \text{if } |\text{caseValue} - \text{queryValue}| > \text{threshold} \\ 1, & \text{otherwise} \end{cases}$$

where *caseValue* is the argument value of the evaluated case, *queryValue* is the argument value of the evaluated situation description and *threshold* is the function parameter.

- The *threshold* field defines the maximal acceptable difference between case value and situation description value.
- **ContextEqual:** This local similarity function behaves exactly as the *Equal* similarity measure defined above except that its result is contextual to the value of a predefined argument of the case and description. If the “context” argument of the case and description are equal, then the *Equal* similarity measure will be performed on the current argument otherwise its value will be set automatically to 0.
  - The *TemplateArgumentReference* which defines an argument of the template that must be equal to the same argument of the description being compared to be evaluated.
- **ContextLessIsBetter:** This local similarity function behaves exactly as the *InrecaLessIsBetter* similarity measure defined above except that its result is contextual to the value of a predefined argument of the case and description. If the “context” argument of the case and description are equal, then the *InrecaLessIsBetter* similarity measure will be performed on the current argument otherwise its value will be set automatically to 0.
  - The *TemplateArgumentReference* which defines an argument of the template that must be equal to the same argument of the description being compared to be evaluated.
  - *maxValue* : see *InrecaLessIsBetter* similarity measure description above.
  - *jump*: see *InrecaLessIsBetter* similarity measure description above.
- **ContextMoreIsBetter:** This local similarity function behaves exactly as the *InrecaMoreIsBetter* similarity measure defined above except that its result is contextual to the value of a predefined argument of the case and description. If the “context” argument of the case and description are equal, then the *InrecaMoreIsBetter* similarity measure will be performed on the current argument otherwise its value will be set automatically to 0.
  - The *TemplateArgumentReference* which defines an argument of the template that must be equal to the same argument of the description being compared to be evaluated.
  - *jump* see *InrecaMoreIsBetter* similarity measure description above.
- **ContextInterval:** This local similarity function behaves exactly as the *Interval* similarity measure defined above except that its result is contextual to the value of a predefined argument of the case and description. If the “context” argument of the case and description are equal, then the *Interval* similarity measure will be

performed on the current argument otherwise its value will be set automatically to 0.

- The *TemplateArgumentReference* which defines an argument of the template that must be equal to the same argument of the description being compared to be evaluated.
- *Interval*: see *Interval* similarity measure description above.
- **Case**: This class defines a template for a given situation template. Cases from the case base will be compared against a situation description to automatically deduce new facts if both are similar.
  - The *id* field uniquely identify the case among all cases of all templates of the context. This id will be set in the inferred fact justification to identify which case was found similar to the input situation.
  - The *description* field is the case description – a list of facts describing a known situation description matching the template.

### 2.5.3 Service endpoints

The Web Service interface endpoint has the following scheme:

`http://host:port/istip/mri-cbr/CaseBasedReasonerWS`

where the *port* is usually 8080 in the case of JBoss

Moreover, the WSDL is available by simply adding “?wsdl” at the end of the previous address scheme when deployed in JBoss.

The Web Service interface endpoint has also been registered to the ISTIP UDDI:

*Table 5: Case-Based Reasoner Service Endpoint*

Service Name	Value
Case Based Reasoner Service	<code>http://10.9.1.200:8080/istip/mri-cbr/CaseBasedReasonerWS</code>

The CBR service is also available through Java remote and local interfaces through JBoss using the following JNDI names:

*Table 6: Case-Based Reasoner Service JNDI Names*

JNDI Name	Interface	Type
<code>MRI/CaseBasedReasonerBean/local</code>	<code>ca.gc.rddc.casebasedreasoner.CaseBasedReasonerLocal</code>	Local
<code>MRI/CaseBasedReasonerBean/remote- ca.gc.rddc.casebasedreasoner.CaseBasedReasonerRemote</code>	<code>ca.gc.rddc.casebasedreasoner.CaseBasedReasonerRemote</code>	Remote



## 2.5.4 Security

The CBR service can be secured through OpenSSO authentication if deployed with ant using the following parameter value:

apply-security = true

Actually, the service is only configured to allow access to any authenticated user when deployed with the above parameter value.

Please refer to the document *Contract Number: W7701-5-3182, Task Authorization 69, SOFTWARE INSTALLATION GUIDE (SIG) – SSO* for more information on how to configure OpenSSO with JBoss.

Please also refer to the document *User Single Sign-on – Development Report – Group C Deliverable 1 – JCDS-CTB-TA69-310-0432-DR* for more information about the service security configuration.

## 2.6 Descriptive Logic Reasoner

### 2.6.1 Methods

- UUID<sup>8</sup> createContext(Collection<FactDTO> facts, Collection<FactDefinitionDTO> atomDefinitions, DescriptiveLogicReasonerKnowHow knowHow, DescriptiveLogicReasonerParameters parameters)

This method allows creating a reasoning context. It is the first step for performing automatic reasoning. It initializes a specific "working context" with the provided facts, atom definitions, knowhow and parameters, and then return the initialized context identifier. This identifier will then be used to perform any other actions on this context.

Please note that this method doesn't launch the reasoning execution. The client must explicitly invoke the *executeAsync* method (or *execute* method for the WS) afterward to do so.

This method may throw a *ReasonerException* if anything goes wrong during the context creation (for example if there is an invalid template in the knowhow).

---

<sup>8</sup> Please note that in the case of the Web Service interface, UUID objects are replaced by Strings, which can be easily transferred using SOAP standard.



## 2.6.2 Service data

### 2.6.2.1 FactDTO and FactDefinitionDTO

These objects are imported directly from the SFM service API containing the fact model. Please refer to the section 3.7 of the document JCDS-CTB-TA69-310-0426-DR-v0.7.doc for more information about these objects.

### 2.6.2.2 DescriptiveLogicReasonerParameters

The following diagram depicts the class structure of the DescriptiveLogicReasonerParameters class:

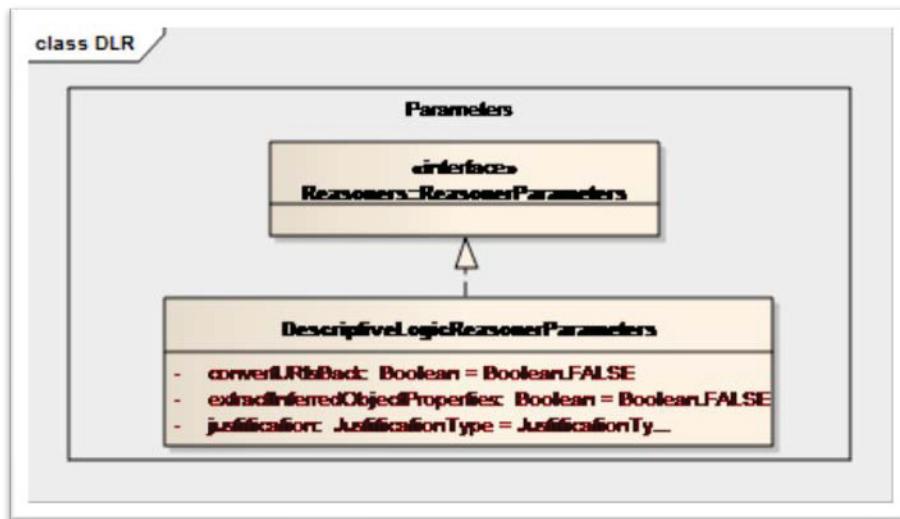


Figure 14: DLR parameters class diagram

Here is a brief description of the classes presented in this diagram:

- **DescriptiveLogicReasonerParameters:** This class defines the general parameters for the descriptive logic reasoner. It only contains one (1) parameter:
  - The *convertURIsBack* field defines if inferred facts URIs will be converted back to the system URIs;
  - The *extractInferredObjectProperties* field defines if object properties inferred will be extracted as facts;
  - The *justification* field defines if the pellet reasoning justifications must be extracted and attached to the generated facts. Be aware that this is a heavy operation that is resource and time consuming;

#### DescriptiveLogicReasonerKnowHow

The following diagram depicts the class structure of the DescriptiveLogicReasonerKnowHow class:

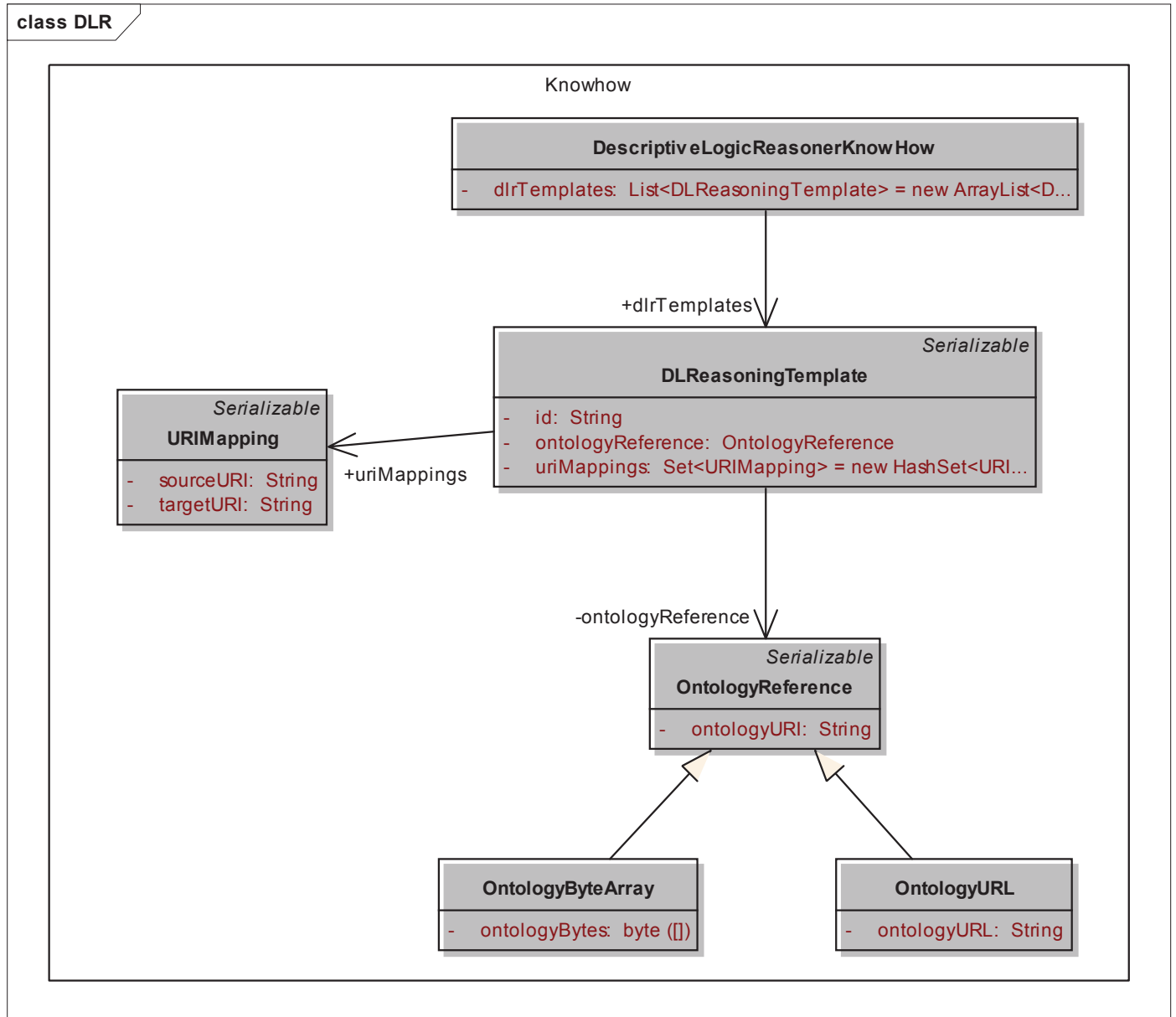


Figure 15: DLR knowhow class diagram

Here is a brief description of the classes presented in this diagram:

- **DescriptiveLogicReasonerKnowHow:** This class basically contains a list of descriptive logic reasoning templates.
- **DLReasoningTemplate:** This class contains an ontology from which new facts will be inferred and a list of URI mapping which can be used to transform triplets into new triplets matching the ontology content.
  - The *id* field uniquely identify the template;

- The *ontologyReference* field specifies which ontology will be used to infer new facts. Its reference can be a byte array, an URL or an URI as described in the *OntologyReference* class below;
- The *uriMappings* field defines a set of *URIMapping* which defines the source URIs to convert to target URIs.
- **URIMapping:** This class defines a source URI and a target URI. During the inference, all triplets matching the source URI will be converted to the target URI before being inserted into the ontology.
  - The *sourceURI* field defines the URI to be translated
  - The *targetURI* field defines the URI the *sourceURI* will be translated to
- **OntologyReference:** This class is used to define an ontology that will be imported from the ontology repository to be processed by the DLR.
  - The *ontologyURI* field is used to import the ontology from the ontology repository
- **OntologyByteArray:** This class is used to define an ontology passed as a byte array to be processed by the DLR.
  - The *ontologyBytes* field contains the ontology content as a byte array.
- **OntologyURL:** This class is used to define an ontology that will be imported from an URL to be processed by the DLR.
  - The *ontologyURL* field defines the URL from which the ontology will be imported.

### 2.6.3 Service endpoints

The Web Service interface endpoint has the following scheme:

`http://host:port/istip/mri-dlr/DescriptiveLogicReasonerWS`

where the *port* is usually 8080 in the case of JBoss

Moreover, the WSDL is available by simply adding “?wsdl” at the end of the previous address scheme when deployed in JBoss.

The Web Service interface endpoint has also been registered to the ISTIP UDDI:

*Table 7: Descriptive Logic reasoner Service Endpoint*

Service Name	Value
Descriptive Logic Reasoner Service	http://10.9.1.200:8080/istip/mri-dlr/DescriptiveLogicReasonerWS

The DLR service is also available through Java remote and local interfaces through JBoss using the following JNDI names:

*Table 8: Descriptive Logic Reasoner Service JNDI Names*

JNDI Name	Interface	Type
MRI/DescriptiveLogicReasonerBean/local	ca.gc.rddc.descriptivelogicreasoner.DescriptiveLogicReasonerLocal	Local
MRI/DescriptiveLogicReasonerBean/remote- ca.gc.rddc.descriptivelogicreasoner.DescriptiveLogicReasonerRemote	ca.gc.rddc.descriptivelogicreasoner.DescriptiveLogicReasonerRemote	Remote

## 2.6.4 Security

The DLR service can be secured through OpenSSO authentication if deployed with ant using the following parameter value:

apply-security = true

Actually, the service is only configured to allow access to any authenticated user when deployed with the above parameter value.

Please refer to the document *Contract Number: W7701-5-3182, Task Authorization 69, SOFTWARE INSTALLATION GUIDE (SIG) – SSO* for more information on how to configure OpenSSO with JBoss.

Please also refer to the document *User Single Sign-on – Development Report – Group C Deliverable 1 – JCDS-CTB-TA69-310-0432-DR* for more information about the service security configuration.

## 2.7 MRI Orchestrator

### 2.7.1 Methods

- UUID<sup>9</sup> createContext(Collection<FactDTO> facts, Collection<FactDefinitionDTO> atomDefinitions, MRIOrchestratorKnowHow knowHow, MRIOrchestratorParameters parameters)

This method allows creating a reasoning context. It is the first step for performing automatic reasoning. It initializes a specific "working context" with the provided facts, atom definitions, knowhow and parameters, and then return the initialized context identifier. This identifier will then be used to perform any other actions on this context.

Please note that this method doesn't launch the reasoning execution. The client must explicitly invoke the *executeAsync* method (or *execute* method for the WS) afterward to do so.

<sup>9</sup> Please note that in the case of the Web Service interface, UUID objects are replaced by Strings, which can be easily transferred using SOAP standard.

This method may throw a *ReasonerException* if anything goes wrong during the context creation (for example if there is an invalid template in the knowhow).

## 2.7.2 Service data

### 2.7.2.1 FactDTO and FactDefinitionDTO

These objects are imported directly from the SFM service API containing the fact model. Please refer to the section 3.7 of the document JCDS-CTB-TA69-310-0426-DR-v0.7.doc for more information about these objects.

### 2.7.2.2 MRIOrchestratorParameters

The following diagram depicts the class structure of the MRIOrchestratorParameters class:

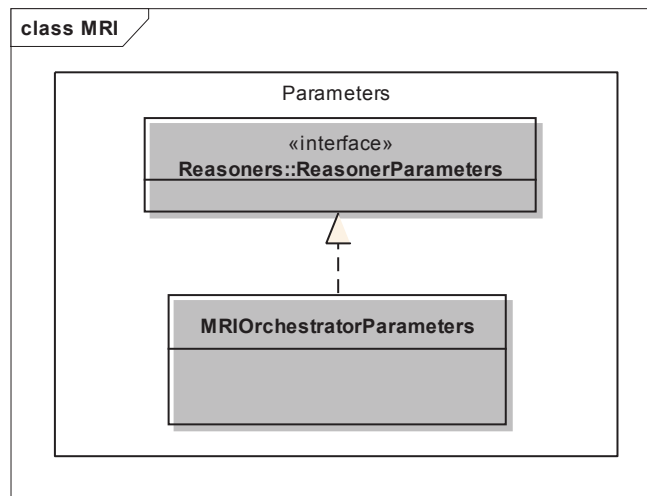


Figure 16: MRI Orchestrator parameters class diagram

Here is a brief description of the classes presented in this diagram:

- **MRIOrchestratorParameters:** This class defines the general parameters for the MRI orchestrator. At the moment it contains no parameters but it has been created to be the placeholder of future parameters.

### 2.7.2.3 MRIOrchestratorKnowHow

The following diagram depicts the class structure of the MRIOrchestratorKnowHow class:

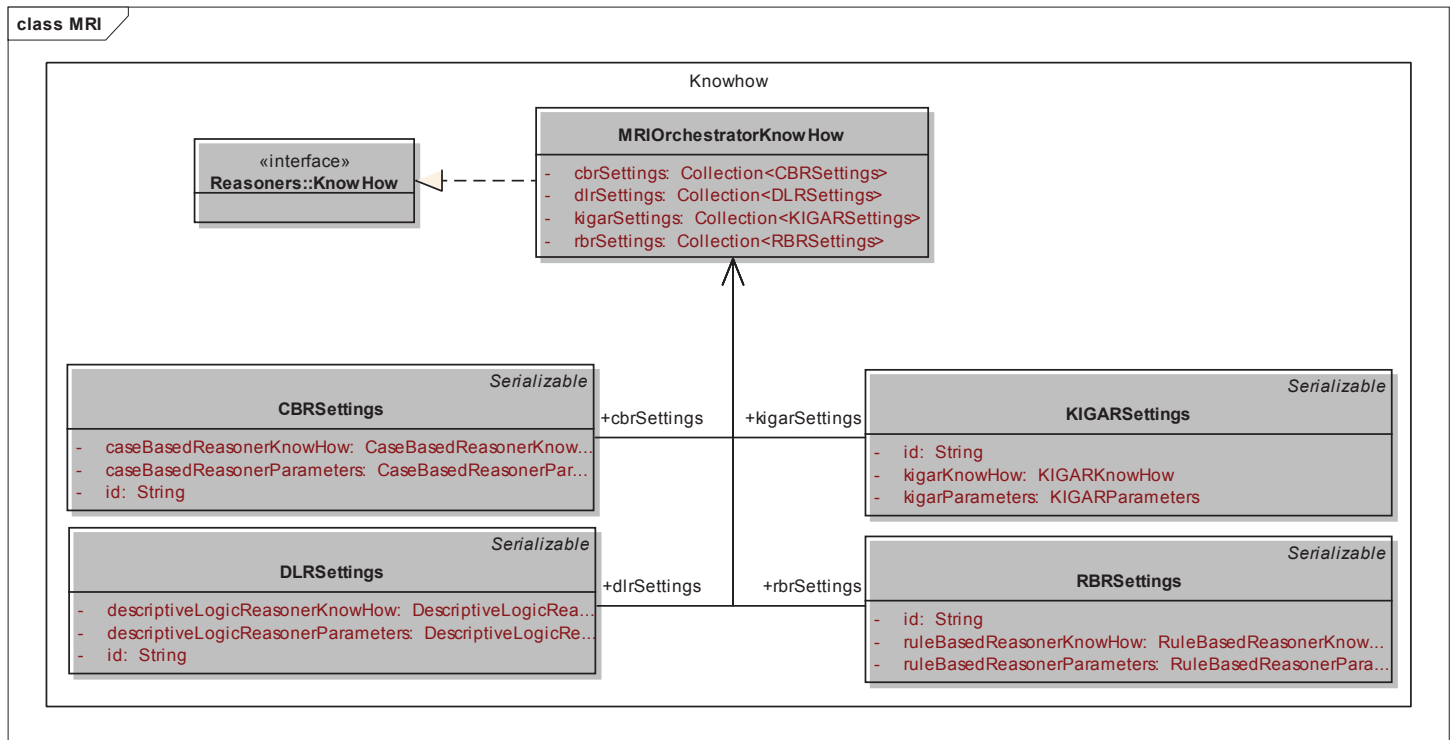


Figure 17: MRI Orchestrator knowhow class diagram

Here is a brief description of the classes presented in this diagram:

- **MRIOrchestratorKnowHow:** This class basically contains reasoners knowhow and parameter for every reasoner that can be called by the MRI. The client must fill only the fields of the reasoners he want to infer with. The fields are:
  - The *cbrSettings* field contains a collection of settings for CBR instances (knowhow + parameters, see CBR section above for more information);
  - The *rbrSettings* field contains a collection of settings for RBR instances (knowhow + parameters, see RBR section above for more information);
  - The *dlrSettings* field contains a collection of settings for DLR instances (knowhow + parameters, see DLR section above for more information);
  - The *kigarSettings* field contains a collection of settings for KIGAR instances (knowhow + parameters, see KIGAR section above for more information);
- **CBRSettings:** This class basically contains one instance of the Case Based Reasoner knowhow and and one instance of its parameters. For each CBRSettings created, an independent instance of the reasoner will be created. The more instance of the settings you create, the more reasoners will try to run in

parallel (Up to the maximum amount defined in the application deployment descriptors<sup>10</sup>).

- **RBRSettings:** This class basically contains one instance of the Rule Based Reasoner knowhow and and one instance of its parameters. For each RBRSettings created, an independent instance of the reasoner will be created. The more instance of the settings you create, the more reasoners will try to run in parallel (Up to the maximum amount defined in the application deployment descriptors<sup>11</sup>).
- **DLRSettings:** This class basically contains one instance of the Descriptive Logic Reasoner knowhow and and one instance of its parameters. For each DLRSettings created, an independent instance of the reasoner will be created. The more instance of the settings you create, the more reasoners will try to run in parallel (Up to the maximum amount defined in the application deployment descriptors<sup>12</sup>).
- **KIGARSettings:** This class basically contains one instance of the KIGAR knowhow and and one instance of its parameters. For each KIGARSettings created, an independent instance of the reasoner will be created. The more instance of the settings you create, the more reasoners will try to run in parallel (Up to the maximum amount defined in the application deployment descriptors<sup>13</sup>).

### 2.7.3 Service endpoints

The Web Service interface endpoint has the following scheme:

`http://host:port/istip/mri-orchestrator/MRIOrchestratorWS`

where the *port* is usually 8080 in the case of JBoss

Moreover, the WSDL is available by simply adding “?wsdl” at the end of the previous address scheme when deployed in JBoss.

The Web Service interface endpoint has also been registered to the ISTIP UDDI:

*Table 9: MRI Orchestrator Service Endpoint*

Service Name	Value
MRI Orchestrator Service	<code>http://10.9.1.200:8080/istip/mri-orchestrator/MRIOrchestratorWS</code>

The DLR service is also available through Java remote and local interfaces through JBoss using the following JNDI names:

<sup>10</sup> See the `ejb-jar.xml` file in the of the “src/META-INF” folder of each project to modify those properties

<sup>11</sup> See the `ejb-jar.xml` file in the of the “src/META-INF” folder of each project to modify those properties

<sup>12</sup> See the `ejb-jar.xml` file in the of the “src/META-INF” folder of each project to modify those properties

<sup>13</sup> See the `ejb-jar.xml` file in the of the “src/META-INF” folder of each project to modify those properties

Table 10: MRI Orchestrator Service JNDI Names

JNDI Name	Interface	Type
MRI/MRIOrchestratorBean/local	ca.gc.rddc.mriorchestrator.MRIOrchestratorLocal	Local
MRI/MRIOrchestratorBean/remot e- ca.gc.rddc.mriorchestrator.MRIO rchestratorRemote	ca.gc.rddc.mriorchestrator.MRIOrchestratorRemote	Remot e

#### 2.7.4 Security

The DLR service can be secured through OpenSSO authentication if deployed with ant using the following parameter value:

apply-security = true

Actually, the service is only configured to allow access to any authenticated user when deployed with the above parameter value.

Please refer to the document *Contract Number: W7701-5-3182, Task Authorization 69, SOFTWARE INSTALLATION GUIDE (SIG) – SSO* for more information on how to configure OpenSSO with JBoss.

Please also refer to the document *User Single Sign-on – Development Report – Group C Deliverable 1 – JCDS-CTB-TA69-310-0432-DR* for more information about the service security configuration.



## 3 Visualization Services

### 3.1 Multi-Reasoners Inference Graphical Visualisation Services (MRIV)

#### 3.1.1 Solution description

The MRIV goal is to add a UI layer on top of various ISTIP services such as the Situational Fact Management service and the Multi-Reasoners Orchestrator service. The first is consumed to perform CRUD operation on situational facts and fact containers, the second is used to interact with the MRI services.

The major goals of the MRIV are to facilitate the user interaction with the Multi-Reasoners Inference service and be able to create or modify a reasoning context, to run a Multi-Reasoners inference process, and to consult the reasoning results.

Finally, the Google Web Toolkit is the platform for remote procedure calls between the client (javascript in web browser) and the server (Java servlets).

Here are links to each' documentation page:

- SmartGWT: <http://code.google.com/p/smartgwt/>
- GWT: <http://code.google.com/webtoolkit/>
- GWT-openlayers: <http://code.google.com/p/gwtopenlayers/>

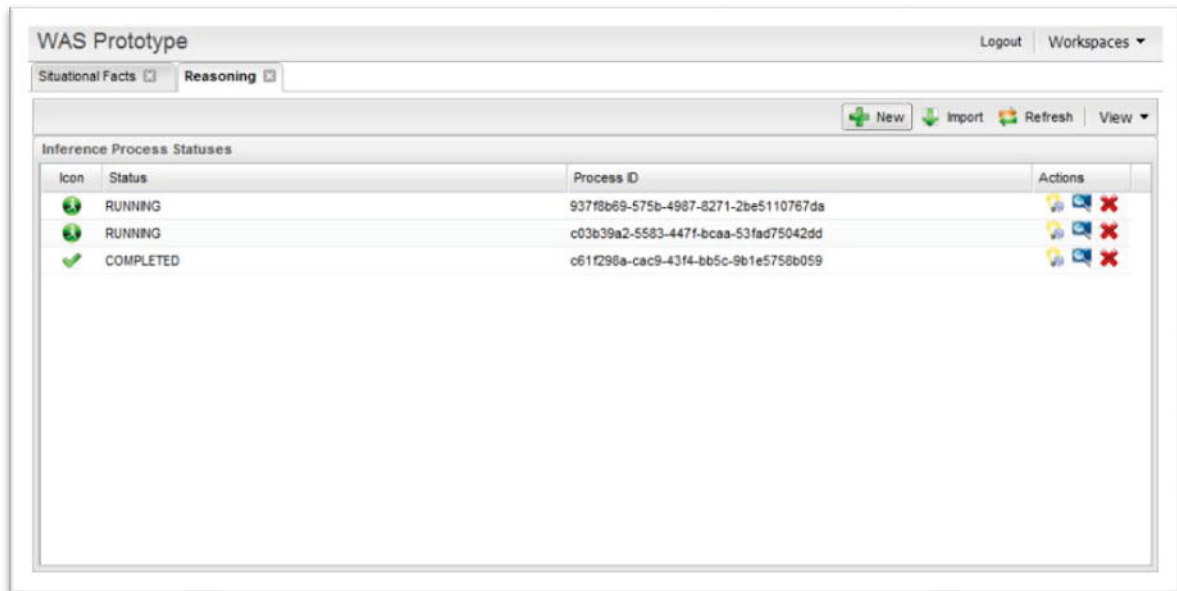


Figure 18: Multi-Reasoner Inference GWT Voila Workspace

### 3.1.1.1 MultiReasonerInferenceGWT project

Multi-Reasoner Inference widgets and services have been developed in MultiReasonerInferenceGWT project. To use these, applications must add the built .jar to their classpath or add a reference to the project itself. Application also have to add an inherit tag in the GWT application module.

```
<inherits name='ca.gc.rddc.was.mri.MultiReasonerInference' />
```

Application must also add the framework module and all data module required in the application.

```
<inherits name='ca.gc.rddc.was.Was' />
```

Moreover since it is depending on the SituationalFactsManagementGWT project, it also requires the importation of the SituationalFactsManagementGWT project:

```
<inherits name='ca.gc.rddc.was.sfm.SituationalFactsManagement' />
```

### 3.1.1.2 Services

#### 3.1.1.2.1 MRIOrchestratorService

MRIOrchestratorService, MRIOrchestratorServiceAsync interfaces and MRIOrchestratorServiceImpl define a GWT remote service that interacts with the MRIOrchestratorService EJB service. It requires the deployment of the MRIOrchestratorService EJB service on the server.

Application's web.xml file must contain the proper servlet definition.

```
<servlet>
  <servlet-name>mriOrchestratorServiceImpl</servlet-name>
  <servlet-class>
    ca.gc.rddc.was.mri.server.MRIOrchestratorServiceImpl
  </servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>mriOrchestratorServiceImpl</servlet-name>
  <url-pattern>/wasprototype/mriOrchestratorService</url-pattern>
</servlet-mapping>
```

MRIOrchestratorService is used in the different reasoning widgets for operations on the MRI Orchestrator processed by the UI layer.

#### 3.1.1.2.2 PredefinedMRIContextService

PredefinedMRIContextService, PredefinedMRIContextServiceAsync interfaces and PredefinedMRIContextServiceImpl define a GWT remote service.

Application's web.xml file must contain the proper servlet definition.

```
<servlet>
  <servlet-name>predefinedMRIContextsServiceImpl</servlet-name>
  <servlet-class>
    ca.gc.rddc.was.mri.server.PredefinedMRIContextsServiceImpl
  </servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>predefinedMRIContextsServiceImpl</servlet-name>
  <url-pattern>/wasprototype/predefinedMRIContextsService</url-
pattern>
</servlet-mapping>
```

The PredefinedMRIContextService is used to extract the predefined MRI contexts stored on the server in the configuration folder so that these predefined contexts can be used in a Multi-Reasoners inference process.




### 3.1.1.3 Widgets and Windows

#### 3.1.1.3.1 Inference Process Dashboard

##### Description

The Inference Process Dashboard is a list grid used to display the created multi-reasoner inference processes. It displays, the process status icon, the process status, the process id and the action that can be accomplished for each of these processes.

These actions are:

- Add facts to an existing process context (  )
- Consult inference results (  )
- Delete the process inference context (  )

To execute these actions click on the corresponding button.

##### Related Classes

ReasoningResultsWindow

ContextEditorWindow

ReasoningResults

##### Screenshot













Inference Process Statuses			
Icon	Status	Process ID	Actions
	RUNNING	937f8b69-575b-4987-8271-2be5110767da	  
	RUNNING	c03b39a2-5583-447f-bcaa-53fad75042dd	  
	COMPLETED	c61f298a-cac9-43f4-bb5c-9b1e5758b059	  

Figure 19: Inference Process Dashboard

### 3.1.1.3.2 Context Editor Window

#### Description

This window contains all the necessary components used to create an inference process context. It contains a fact creation component, a “parameters and knowhow” selection component and a situational fact container selector.

In the *Input facts* section, the user can select an atom definition and fill its arguments to create a fact that will be injected in the inference process context when the user will click the create context button.

In the *Parameters and Knowhow* section, the user must select a single context file containing the set of parameters and knowhow to pass to the inference process.

In the *Additional situational facts* section, the user can select one or more fact container to add situational facts to the reasoning context.

Finally to create the context, the user must press the *Create context* button or the *Cancel* button to cancel the inference process context creation.

#### Related Class

Atom Definition

Parameters

KnowHow

FactContainer

Fact

#### Screenshot

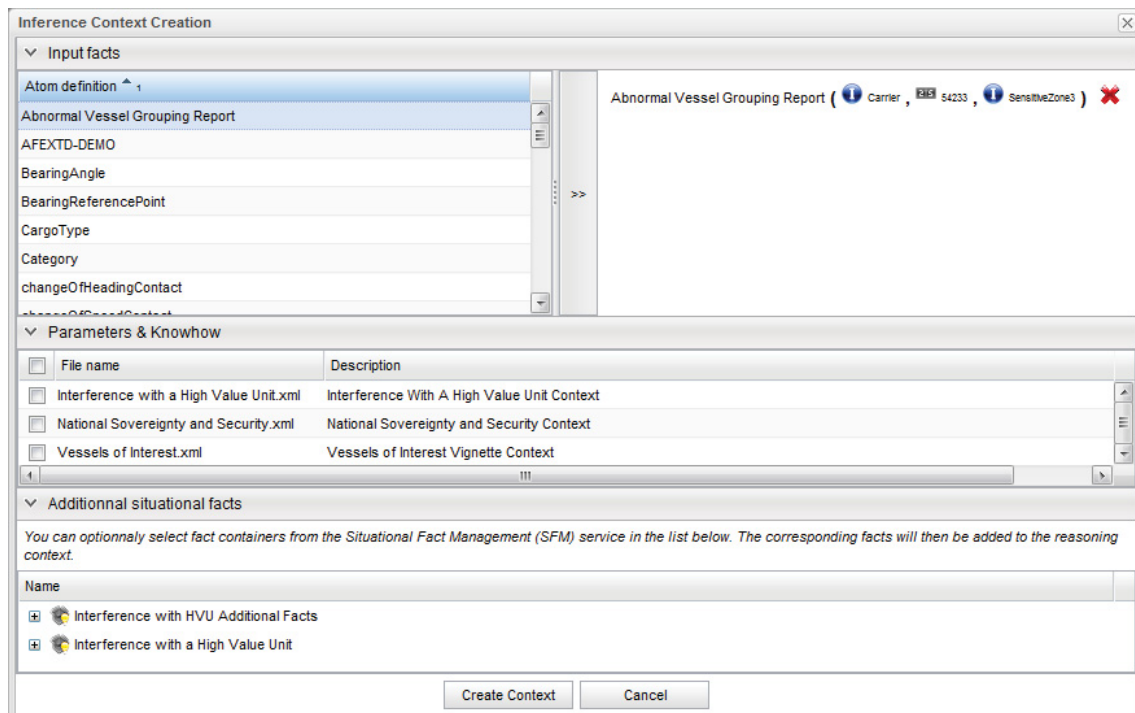


Figure 20: Context Editor Window

### 3.1.1.3.3 Reasoning Results Window

#### Description

The reasoning results window is used to display inferred facts to the user. It contains a list of inferred facts with their pedigree.

Available columns are:

- Fact Id
- Fact
- Fact Pedigree
- Fact Validity
- Fact Confidence
- Fact Classification

When double clicking on the fact, the user can navigate through the fact attributes through the use of the SFM FactAttributesWidget.

The *Messages* tab of this window displays messages returned by the system.

#### Related Classes

FactAttributesWidget

Attribute

Fact

## Screenshot

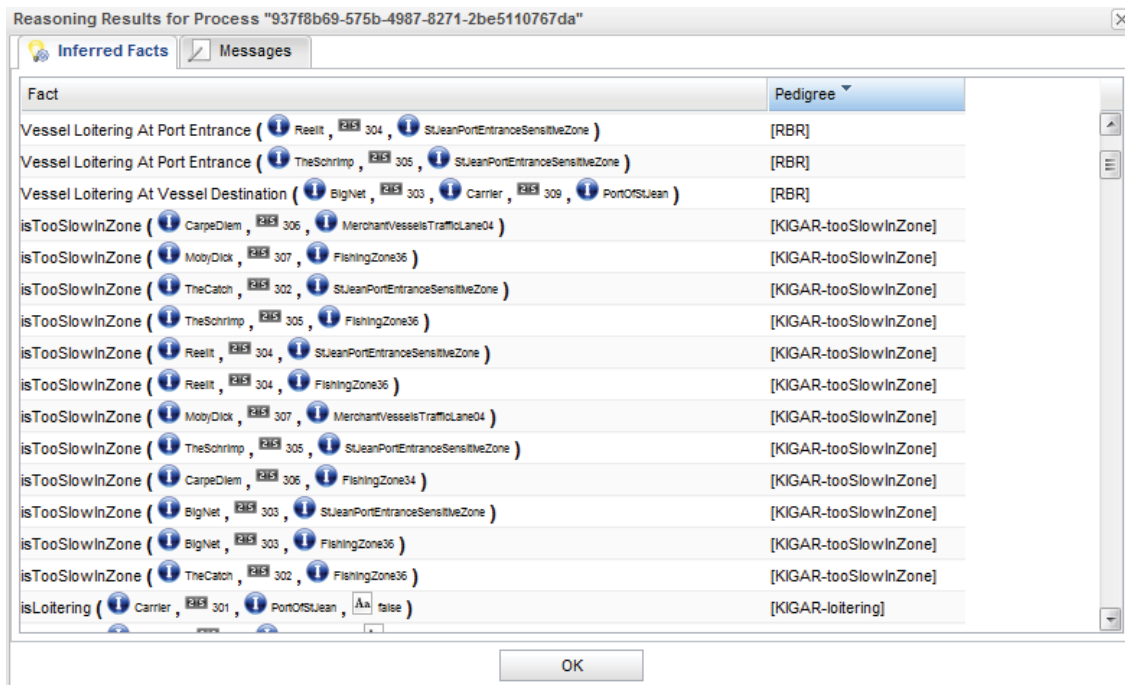


Figure 21: Reasoning Results Window

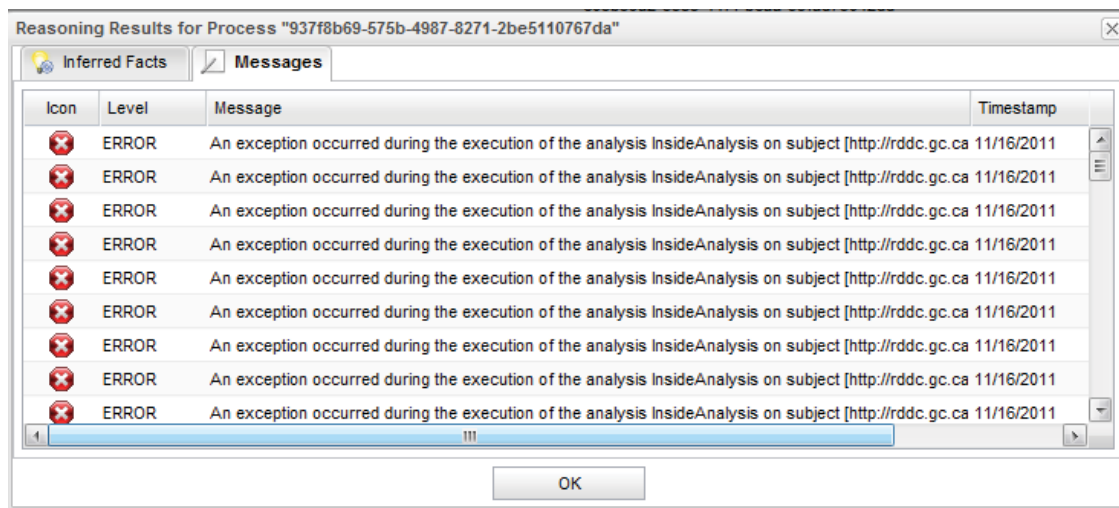





Figure 22: Reasoning Results Window 2

### 3.1.1.3.4 Multi Reasoner Inference Workspace Tool Strip

#### Description

This widget displays button representing the different actions that can be executed anytime in the multi-reasoner inference workspace.

The actions are:

- Create a new inference process (  New )
- Import an existing inference process id to monitor it (  Import )
- Refresh the inference process dashboard statuses (  Refresh )

### Related Classes

InferenceProcessDashboard

ReasoningResults

### Screenshot



Figure 23: Multi Reasoner Inference Workspace Tool Strip

#### 3.1.1.4 Data Modules

The MultiReasonerInferenceGWT project defines modules to make the data transfer objects of the ReasonersCommon, CaseBasedReasonerAPI, RuleBasedReasonerAPI, DescriptiveLogicReasonerAPI, KIGARAPI and MRIOrchestratorAPI available in GWT applications. Thus, a \*.gwt.xml file has been created in the packages corresponding to these projects namespaces to include sources of objects from these projects. To make it work, the main module (MultiReasonerInference.gwt.xml) has six "inherits" tags to include these data modules:

```
<inherits name='ca.gc.rddc.mriorchestrator.MRIOrchestrator' />
<inherits name='ca.gc.rddc.reasonerscommon.ReasonersCommon' />
<inherits name='ca.gc.rddc.kigar.KIGAR' />
<inherits name='ca.gc.rddc.descriptivelogicreasoner.DLR' />
<inherits name='ca.gc.rddc.rulebasedreasoner.RBR' />
<inherits name='ca.gc.rddc.casebasedreasoner.CBR' />
```

#### 3.1.2 Bug report

No formal QA has been applied on the project. Bugs were found and fixed during the development process so no reports are available.

#### 3.1.3 Installation guide

To build the WAS Prototype:

```
ant -f build.xml build.wasprototype.all
```

To install WAS Prototype:

```
ant -f build-install.xml deploy.was.prototype.service
```

#### **3.1.4 Testing procedure and test results**

No functional testing has been performed other than human testing. Creation of functional UI test are possible but have not been created in the scope of this project.



## References

---

- [1] **Vincent Giroux, Guillaume Morin-Brassard – Fujitsu Consulting Canada Inc.** *Multi-Reasoners Inference Software Architecture Description*. Quebec (for DRDC Valcartier), 2011
- [2] **Fujitsu Consulting Canada Inc.** Development Report – Group A Deliverable 3 – JCDS-CTB-TA69-310-0426-DR. Quebec (for DRDC Valcartier), 2011
- [3] **Yannick Allard - OODA Technologies Inc.** *Kinematic and Geospatial Analysis Module (KIGAM) Analysis Fact Sheets*. Montreal (for DRDC Valcartier), 2011
- [4] **Walsh David; Bouchard Kevin; Roy, Jean (Scientific Authority)**. Contract Number: W7701-5-3182, Task Authorization 69, SOFTWARE INSTALLATION GUIDE (SIG) - SSO. Quebec : DMR Conseil, 2011
- [5] **Kevin Bouchard – Fujitsu Consulting Canada**. User Single Sign-on – Development Report – Group C Deliverable 1 – JCDS-CTB-TA69-310-0432-DR. Quebec (for DRDC Valcartier), 2011

This page intentionally left blank.

## Annex A KIGAR Analyses

The following table lists the different analyses available within KIGAR, their parameters, required subjects and objects, as well as the conclusion facts they generate. Please refer to sections 5 and 6 of the *Kinematic and Geospatial Analysis Module (KIGAM) Analysis Fact Sheets* document for more information. The analysis descriptions have been imported from this document.

Table 11: KIGAR Analyses

Analysis	Parameters	Subject type <sup>14</sup>	Object type <sup>15</sup>	Conclusion
<b>PROXIMITY</b> <i>This analysis will verify if a track is within a certain distance of a spatial feature. The last contact of the track will be used as the reference point for the analysis.</i>	<b>proximityThreshold</b> (default: 5000.0) <i>The distance threshold to consider that two spatial features are in proximity (in meters)</i>	1	2	<i>inProximity(subjectUri, subjectTrajectoryId, objectUri)</i>
<b>INSIDE</b> <i>This analysis will verify if a track is contained inside a Spatial Feature. The last contact of the track will be used as the reference point for the analysis.</i>	<i>None</i>	1	2	<i>isInside(subjectUri, subjectTrajectoryId, subjectLastContactId, objectUri)</i>
<b>OUTSIDE</b> <i>This analysis will verify if a track is not contained inside a Spatial Feature. The last contact of the track will be used as the reference point for the analysis.</i>	<i>None</i>	1	2	<i>isOutside(subjectUri, subjectTrajectoryId, subjectLastContactId, objectUri)</i>
<b>MANEUVER</b> <i>This analysis will verify if there is a significant change of heading or change of speed inside the track. Each segment from the track model will be used to compare with the threshold.</i>	<b>maneuverCourseChangeLimit</b> (default: 20.0) <i>The change of heading threshold (in degrees) before identifying significant maneuver change contacts</i> <b>maneuverSpeedChangeLimit</b> (default: 5.0) <i>The change of speed threshold (in meters/sec) before identifying significant maneuver change contacts</i>	1	N/A	<i>maneuverContact(subjectUri, subjectTrajectoryId, subjectContactId)</i>
<b>CHANGE OF HEADING</b> <i>This analysis will verify if there is a significant change of heading inside the track. Each segment from the track model will be used to compare with the threshold.</i>	<b>cohCourseChangeLimit</b> (default: 20.0) <i>The change of heading threshold (in degrees) before identifying major change of heading contacts</i>	1	N/A	<i>changeOfHeadingContact(subjectUri, subjectTrajectoryId, subjectContactId)</i>
<b>CHANGE OF SPEED</b> <i>This analysis will verify if there is a significant change of speed inside the track. Each segment from the track model will be used to compare with the threshold.</i>	<b>cosSpeedChangeLimit</b> (default: 5.0) <i>The maximum allowed speed (in meters/sec) change between two contacts before adding a potential anomaly indication</i>	1	N/A	<i>changeOfSpeedContact(subjectUri, subjectTrajectoryId, subjectContactId)</i>

<sup>14</sup> The numbers in this column determine which kinds of subjects are supported by each analysis. Indeed, subjects will be filtered first by making sure a “is-a” triplet associating the subject with the type URI is present within the input facts triplets. The list of subjects/objects types is available below this table.

<sup>15</sup> The numbers in this column determine which kinds of objects are supported by each analysis. Objects will be filtered first by making sure a “is-a” triplet associating the object with the type URI is present within the input facts triplets. The list of subjects/objects types is available below this table.

<b>RENDEZVOUS</b> <i>This analysis will verify if there is a significant chance that two tracks might have been at the same position at the same time based on the modeled trajectories.</i>	<b>rvTimeTolerance</b> (default: 3600.0) <i>The time threshold (in seconds) separating the two spatio-temporal features</i> <b>rvGeographicScore</b> (default: 0.5) <i>The geospatial feasibility score threshold</i>	1	3	<i>hasRendezVous(subjectUri, subjectTrajectoryId, objectUri, objectTrajectoryId)</i>
<b>STOP_EMITTING</b> <i>This analysis will verify if there is a track that has stop emitting, i.e. reporting its position.</i>	<b>seProximityThreshold</b> (default: 5000.0) <i>The distance threshold to consider that two spatial features are in proximity (in meters)</i> <b>seTimeThreshold</b> (default: 3600) <i>The time threshold (in seconds) between the reference time and the subject's last contact time</i> <b>seReferenceTime</b> (default: current time) <i>The reference time to compare with the subject's last contact time (in milliseconds since January 1, 1970, 00:00:00 GMT). If not set, the current date is used</i>	1	4	<i>stoppedEmitting(subjectUri, subjectTrajectoryId)</i>
<b>APPARENT_DESTINATION</b> <i>This analysis will derive the possible destinations of a track based on the estimated time of arrival (ETA)</i>	<b>adCourseDeviation</b> (default: 15.0) <i>The maximum allowed course deviation (in degrees)</i>	1	2	<i>hasApparentDestination(subjectUri, subjectTrajectoryId, destinationUri)</i>
<b>COURSE_PROXIMITY</b> <i>This analysis will determine if tracks have course proximity based on the specified proximity threshold. Note that the computed distance will be the smallest along the entire trajectory of the tracks, which is internally represented as a curve object and not only between inflection points of those curves.</i>	<b>cpTargetProximityThreshold</b> (default: 5000.0) <i>The distance threshold to consider that two spatial features are in proximity (in meters)</i>	1	3	<i>hasCourseProximity(subjectUri, subjectTrajectoryId, objectUri, objectTrajectoryId)</i>
<b>GEOFEASIBLE</b> <i>This analysis will derive the possible geofeasible objects for a track based on its current speed</i>	<b>gfExtrapolationDeltaTime</b> (default: 0.0) <i>The extrapolation time (in milliseconds since January 1, 1970, 00:00:00 GMT) used to analyse if a subject is heading toward an object</i>	1	2	<i>isGeofeasible(subjectUri, subjectTrajectoryId, objectUri)</i>
<b>HEADING_TOWARDS</b> <i>This analysis will determine if a track is heading towards a spatial feature within a given amount of time. It generates a pie slice, based on track's speed and possible course deviation, to determine an area of possible positions. A track is considered to be heading towards all spatial features within this pie slice.</i>	<b>headingCourseDeviation</b> (default: 15.0) <i>The maximum tolerated course deviation (in degrees) to consider that a subject is heading toward an object</i> <b>headingExtrapolationDeltaTime</b> (default: 0.0) <i>The extrapolation time (in milliseconds) used to analyse if a subject is heading toward an object</i>	1	2	<i>isHeadingTowards(subjectUri, subjectTrajectoryId, objectUri)</i>
<b>HEADING_AWAY</b> <i>This analysis will determine if a track is heading away from spatial feature. It generates a pie slice, based on track's speed and possible course deviation, to determine an area of possible positions. A track is considered to be heading away from all spatial features not contained in this pie slice.</i>	<b>headingCourseDeviation</b> (default: 15.0) <i>The maximum tolerated course deviation (in degrees) to consider that a subject is heading away from an object</i> <b>headingExtrapolationDeltaTime</b> (default: 0.0) <i>The extrapolation time (in milliseconds) used to analyse if a subject is heading away from an object</i>	1	2	<i>isHeadingAwayFrom(subjectUri, subjectTrajectoryId, objectUri)</i>
<b>TOO_SLOW</b> <i>This analysis will determine if a track is going too slow in a particular Zone.</i>	<b>zoneMinimumSpeed</b> (default: null → The zone specific minimum speed) <i>The subject's minimum speed (in meters/sec). If not defined, the object's (zone) minimum speed will be used</i>	1	5	<i>isTooSlowInZone(subjectUri, subjectTrajectoryId, zoneUri)</i>
<b>TOO_FAST</b> <i>This analysis will determine if a track is going too</i>	<b>zoneMaximumSpeed</b> (default: null → The zone specific maximum speed)	1	5	<i>isTooFastInZone(subjectUri, subjectTrajectoryId, zoneUri)</i>

<i>fast in a particular Zone.</i>	<i>The subject's maximum speed (in meters/sec). If not defined, the object's (zone) maximum speed will be used</i>			<i>zoneUri)</i>
<b>APPROACHING</b>  <i>This analysis will determine if a track is approaching from a spatial feature given that it is within a certain distance of it.</i>	<i>The minimum distance (in meters) between the subject and the object before considering if the subject is approaching the object</i> <b>distanceThreshold</b> (default: 500000.0)  <i>The time range (in seconds) used to estimate the future subject location based on its current heading and speed.</i> <b>traveledDistanceMultiplier</b> (default: 600.0)	1	2	<i>isApproaching(subjectUri, subjectTrajectoryId, objectUri)</i>
<b>MOVING_AWAY</b>  <i>This analysis will determine if a track is moving away from a spatial feature given that it is within a certain distance of it.</i>	<i>The minimum distance (in meters) between the subject and the object before considering if the subject is moving away from the object</i> <b>distanceThreshold</b> (default: 500000.0)  <i>The time range (in seconds) used to estimate the future subject location based on its current heading and speed.</i> <b>traveledDistanceMultiplier</b> (default: 600.0)	1	2	<i>isMovingAwayFrom(subjectUri, subjectTrajectoryId, objectUri)</i>
<b>CROSSING</b>  <i>This analysis will determine if a track is crossing a spatial feature of type LINEAR FEATURE (e.g. boundary)</i>	<i>None</i>	1	6	<i>isCrossing(subjectUri, subjectTrajectoryId, objectUri)</i>
<b>FOLLOWING</b>  <i>This analysis determines if a track is following a spatial feature of type ROUTE.</i>	<i>The time range (in seconds) since the last subject contact that will be considered by the analysis</i> <b>frTimeBackward</b> (default: 7200.0)  <i>Number of point on the specified time range that will be generated for the analysis, should be high enough to enable precise computation.</i> <b>frNumberOfPoints</b> (default: 10.0)	1	7	<i>isFollowing(subjectUri, subjectTrajectoryId, objectUri)</i>
<b>DEVIATING</b>  <i>This analysis determines if a track is deviating from a spatial feature of type ROUTE.</i>	<i>The time range (in seconds) since the last subject contact that will be considered by the analysis</i> <b>dfrTimeBackward</b> (default: 7200.0)  <i>Number of point on the specified time range that will be generated for the analysis, should be high enough to enable precise computation.</i> <b>dfrNumberOfPoints</b> (default: 10.0)	1	7	<i>isDeviatingFrom(subjectUri, subjectTrajectoryId, objectUri)</i>
<b>LOITERING</b>  <i>This analysis will determine if a track is loitering based on the supplied minimal speed and its destination. If a track has no destination, the analysis will not be applied.</i>	<i>The speed (in knots/sec) at which we can consider that the subject is loitering</i> <b>loiteringKnotsPerSeconds</b> (default: 2.0)  <i>The minimum distance from which a track must be from its destination to be considered loitering</i> <b>loiteringProximityThreshold</b> (default: 1000.0)	1	4	<i>isLoitering ( subjectUri, subjectTrajectoryId, destinationUri, true/false)</i>
<b>INBOUND</b>  <i>This analysis determines if a ship is heading inbound a zone or a country of interest.</i>	<i>The maximum course difference in degree between the heading and the heading to reach feature centroid</i> <b>inboundCourseDifferenceThreshold</b> (default: 20.0)  <i>The distance threshold to consider that two spatial features are in proximity (in meters)</i> <b>inboundProximityThreshold</b> (default: 5000.0)  <b>inboundTraveledDistanceMultiplier</b> (default: 600.0)  <i>The time range (in seconds) used to estimate the future</i>	1	5 or 8	<i>isInbound(subjectUri, subjectTrajectoryId, objectUri)</i>

<b>OUTBOUND</b>  <i>This analysis determines if a ship is heading outbound a zone or a country of interest.</i>	<i>subject location based on its current heading and speed.</i>			
	<b>inboundCourseDifferenceThreshold</b> (default: 20.0)	1	5 or 8	<i>isOutbound(subjectUri, subjectTrajectoryId, objectUri)</i>
	<i>The maximum course difference in degree between the heading and the heading to reach feature centroid</i>			
	<b>inboundProximityThreshold</b> (default: 5000.0)			
	<i>The distance threshold to consider that two spatial features are in proximity (in meters)</i>			
<b>GOING_AROUND</b>  <i>This analysis will determine if a track is going around a particular zone.</i>	<b>inboundTraveledDistanceMultiplier</b> (default: 600.0)			
	<i>The time range (in seconds) used to estimate the future subject location based on its current heading and speed.</i>			
	<b>gaCourseChangeLimit</b> (default: 20.0)	1	5	<i>isGoingAround(subjectUri, subjectTrajectoryId, objectUri)</i>
	<i>The course change threshold (in degrees) for considering that the subject is actually avoiding the object</i>			
	<b>gaProximityThreshold</b> (default: 5000.0)			
<b>GEOFEASIBLE_DESTINATION</b>  <i>This analysis determines if the destination of a track is geofeasible based on its Estimated Time of Arrival (ETA).</i>	<i>The minimum distance (in meters) between the subject and the object for considering that the change of heading is due to the object proximity</i>			
	<b>None</b>	1	4	<i>isDestinationGeofeasible(subjectUri, subjectTrajectoryId, destinationUri, true/false)</i>
<b>ENTERING</b>  <i>This analysis determines if a track is entering a zone.</i>	<b>None</b>	1	5	<i>isEnteringZone(subjectUri, subjectTrajectoryId, zoneUri)</i>
<b>LEAVING</b>  <i>This analysis determines if a track is leaving a zone.</i>	<b>None</b>	1	5	<i>isLeavingZone(subjectUri, subjectTrajectoryId, zoneUri)</i>
<b>PASSING_THROUGH</b>  <i>This analysis will determine if a track is passing through a zone.</i>	<b>None</b>	1	5	<i>isPassingThroughZone(subjectUri, subjectTrajectoryId, zoneUri)</i>
<b>CIRCLE</b>  <i>This analysis determines if a track has a circular motion pattern.</i>	<b>circleCourseChangeLimit</b> (default: 20.0)	1	N/A	<i>isCircling(subjectUri, subjectTrajectoryId)</i>
	<i>The change of heading threshold (in degrees) for selecting the main significant contacts for the analysis</i>			
	<b>circleBias</b> (default: 900.0)			
	<i>The time range buffer (in seconds) allowed to determine, when added to the actual course time, if the course was performing a feasible circle</i>			
<b>STOPPED_AT_DESTINATION</b>  <i>This analysis determines if a track is stopped at its destination</i>	<b>stoppedAtDestinationProximityThreshold</b> (default: 2000.0)	1	N/A	<i>isStoppedAtDestination(subjectUri, subjectTrajectoryId, destinationUri, true/false)</i>
	<i>The maximum distance in meters to consider the track is at destination</i>			
	<b>stoppedAtDestinationKnotsPerSecondThreshold</b> (default: 2.0)			
	<i>The maximum speed in nautical miles to consider the track is stopped.</i>			

\*Subjects/Objects types:

1. <http://localhost:8080/ISFAR/ontologies/SpatialFeatures.owl#Track>
2. <http://localhost:8080/ISFAR/ontologies/SpatialFeatures.owl#StaticSpatialFeature>
3. <http://localhost:8080/ISFAR/ontologies/SpatialFeatures.owl#SpatioTemporalFeature>
4. <http://localhost:8080/ISFAR/ontologies/SpatialFeatures.owl#Destination>
5. <http://localhost:8080/ISFAR/ontologies/SpatialFeatures.owl#Zone>
6. <http://localhost:8080/ISFAR/ontologies/SpatialFeatures.owl#LinearFeature>
7. <http://localhost:8080/ISFAR/ontologies/SpatialFeatures.owl#Route>
8. <http://localhost:8080/ISFAR/ontologies/SpatialFeatures.owl#Country>

This page intentionally left blank.



## List of symbols/abbreviations/acronyms/initialisms

---

CBR	Case-based reasoner
DLR	Descriptive Logic Reasoner – A reasoner based on ontological reasoning.
DTO	Data Transport Object
KIGAM	KInematic and Geospatial Analysis Module
KIGAR	KInematic and Geospatial Analysis Reasoner
MRI	Multi-Reasoner Inference
RBR	Rule-Based Reasoner
URI	Uniform Resource Identifier
WS	Web Service
WSDL	Web Service Definition Language

This page intentionally left blank.

DOCUMENT CONTROL DATA		
(Security markings for the title, abstract and indexing annotation must be entered when the document is Classified or Designated)		
1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)  <b>Fujitsu Consulting (Canada) Inc.</b> <b>2000 Boulevard Lebourgneuf,</b> <b>Bureau 300, Québec (Québec) G2K 0E8</b>		2a. SECURITY MARKING (Overall security marking of the document including special supplemental markings if applicable.)  <b>UNCLASSIFIED</b>
		2b. CONTROLLED GOODS  <b>(NON-CONTROLLED GOODS)</b> <b>DMC A</b> <b>REVIEW: GCEC APRIL 2011</b>
3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)  <b>Multi-reasoner Inference : Software Interface Design Description</b>		
4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)  <b>Morin-Brassard, G.; Giroux, V.</b>		
5. DATE OF PUBLICATION (Month and year of publication of document.)  <b>January 2012</b>	6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.)  <b>70</b>	6b. NO. OF REFS (Total cited in document.)  <b>5</b>
7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)  <b>Contract Report</b>		
8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)  <b>Defence Research and Development Canada – Valcartier</b> <b>2459 Pie-XI Blvd North</b> <b>Quebec (Quebec)</b> <b>G3J 1X5 Canada</b>		
9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)  <b>W7701-10-4064</b>	
10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)  <b>MRI-242-0449</b>	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)  <b>DRDC Valcartier CR 2012-003</b>	
11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)  <b>Unlimited</b>		
12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.)  <b>Unlimited</b>		

13. **ABSTRACT** (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

To support its research activities in the intelligence domain, the Intelligence and Information (I&I) Section at DRDC Valcartier is developing the Intelligence Science & Technology Platform (ISTIP) as a major component of its R&D infrastructures. To improve the reasoning capabilities of the platform, the mandate of this contract is to produce a Multi-Reasoner Inference (MRI) capability based on the Multi-Intelligence Tool Suite (MITS) and the ISTIP software components previously developed by the I&I Section. Five main different services have been developed containing four individual reasoners and one multi-reasoner orchestrator. The reasoners that have been created are a Case-Based Reasoner (CBR), a Rule-Based Reasoner (RBR), a Descriptive-Logic Reasoner (DLR) and a Kinematics and Geospatial Analysis Reasoner (KIGAR) which is based on the KIGAM module of the Inference of Situational Facts through Automated Reasoning (ISFAR) tool. Through the use of a common reasoning framework, these reasoners can now leverage their reasoning capabilities by sharing their strength to other reasoners and achieve an amazing synergy. This document describes the Software Interface Design Description of the MRI.

-----

Afin de supporter ces activités de recherche dans le domaine du renseignement, la Section du Renseignement et Information de RDDC Valcartier développe la Plate-forme de Science et Technologie du Renseignement (ISTIP) comme un composant majeur de ses infrastructures de R&D. Afin d'améliorer les aptitudes de raisonnement de la plate-forme, le mandat de ce contrat est de créer un outil d'inférence Multi-Raisonneur (MRI) basé sur la « Multi-Intelligence Tool Suite » (MITS) et sur les composants logiciels déjà implémentés par la section I&I. Cinq différents services ont été développés comprenant quatre raisonneurs individuels et un orchestrateur multi-raisonneur. Les raisonneurs qui ont été créés sont un raisonneur par cas (CBR), un raisonneur par règles (RBR), un raisonneur ontologique (DLR) et un raisonneur d'analyse cinématique et géo-spatiale (KIGAR) basé sur le module KIGAM de l'outil d'Inférence Automatisée de Faits Situationnels (ISFAR). Grâce à l'utilisation d'un cadre de raisonnement commun, ces raisonneurs peuvent désormais exploiter leurs capacités de raisonnement en partageant leurs forces à d'autres raisonneurs et parvenir à une synergie épatante. Ce document décrit l'Architecture Logicielle des Interfaces du MRI.

14. **KEYWORDS, DESCRIPTORS or IDENTIFIERS** (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

reasoner; inference; Intelligence and Information (I&I) Section; R&D infrastructure; Multi-Reasoner Inference (MRI); Case-Based Reasoner (CBR); Rule-Based Reasoner (RBR); Descriptive-Logic Reasoner (DLR); Kinematics and Geospatial Analysis Reasoner (KIGAR); KIGAM module; Inference of Situational Facts through Automated Reasoning (ISFAR); software interface design description



## **Defence R&D Canada**

Canada's Leader in Defence  
and National Security  
Science and Technology

## **R & D pour la défense Canada**

Chef de file au Canada en matière  
de science et de technologie pour  
la défense et la sécurité nationale



**[www.drdc-rddc.gc.ca](http://www.drdc-rddc.gc.ca)**