\

GUIA DE UTILIZAÇÃO DE MOTOR DE SCRIPTING SOFIA2

JULHO 2014

Versão 4

1 ÍNDICE

2 INTRODUÇÃO

2.1 Requisito

Antes de prosseguir com este guia, recomenda-se a leitura do guia **SOFIA2-Conceitos SOFIA2.doc**

2.2 Objetivos e âmbito do presente documento

O presente documento descreve como utilizar o motor de scripting de SOFIA2, que permite aos utilizadores da plataforma criar os seus próprios scripts, que serão executados antes da chegada de uma instância de ontologia ou então com regularidade.

2.3 Conceitos

- **Tipos de scripts:** a plataforma suporta 2 tipos de scripts:
    - **Scripts mediante eventos**: permite executar o código antes da chegada de instâncias de ontologias
    - **Scripts temporizados:** permite executar o código em determinado tempo, este tempo pode definir-se com um cron.
- **Linguagem de scripting:** os scripts definem-se na linguagem Groovy (**http://groovy.codehaus.org**)
- **APIs disponíveis para os scripts (Operações):** os scripts podem aceder a um conjunto de APIs que permitem executar ações como:
    - Enviar e-mail
    - Invocar URL
    - Inserir instância de Ontologia
    - Consultar instâncias de uma Ontologia
    - Apagar instância de Ontologia

  Os utilizadores com permissões poderão criar as suas próprias APIs.
- **Segurança nos scripts:** os scripts executam-se em contexto seguro, o que garante que um erro num script não afete os restantes. Além disso, têm um tempo de *timeout* na invocação. Os scripts podem apenas invocar um conjunto de operações.

3 DESENVOLVIMENTO DE SCRIPTS

3.1 Groovy

Os scripts desenvolvem-se na linguagem Groovy, que permite programar numa linguagem simples e não tipada.

A linguagem Groovy compila-se na execução de Java e é executada na JVM. A partir desta pode aceder a todas as classes e bibliotecas Java.

Pode obter mais informação sobre a linguagem Groovy em **http://groovy.codehaus.org**.

3.2 API Operações.

As **Operações** permitem definir as APIs acessíveis aos scripts, são os comandos que poderão executar os nossos Scripts Groovy.

As operações podem apenas ser definidas pelos utilizadores administradores.

Segue um exemplo de API:

```
import com.indra.jee.arq.spring.sofia2.script.api.Api;

public class ApiGroovy {

	public class ScriptException extends Throwable{

		public ScriptException(Exception e){
			super(e);
		}

		public ScriptException(Exception e, String message){
			super(message, e);
		}

		public ScriptException(String message){
			super(message);
		}
	}

	private Api api;

	public ApiGroovy(){
		api = new Api();
	}

	public String insert(String ontologyName, String ontology) throws ScriptException{
		try{
			return api.insert(ontologyName, ontology);
		}catch(Exception e){
			throw new ScriptException(e);
		}
	}

	public String rolback(String ontologyName, String id) throws ScriptException{
		try{
			return api.rolback(ontologyName, id);
		}catch(Exception e){
			throw new ScriptException(e);
		}
	}
```

Esta API distribui-se com a plataforma SOFIA2:

- Não tem nenhum pacote definido, já que os Scripts não podem importar Classes, as nossas operações deverão alojar-se no pacote base.
- Realiza a importação de uma Classe, as operações não têm restrições e ao utilizar a Groovy integra-se 100% com java.

```
import com.indra.jee.arq.spring.sofia2.script.api.Api;
```

- Podem definir classes, subclasses e devem definir-se métodos que exponham as operações que queremos utilizar.

```
public class ApiGroovy {
	public class ScriptException extends Throwable{
		public ScriptException(Exception e){
			super(e);
		}
```

3.2.1 ApiGroovy

A operação `ApiGroovy` é uma fachada para a classe API Java, que oferece estes métodos:

- **Insert** insere uma instância de uma ontologia na base de dados em tempo real.
- **Rolbak** elimina uma ontologia da base de dados em tempo real.
- **getAtribute** obtém o valor de um atributo de uma ontologia.
- **sendMail** permite enviar um e-mail.

```
import com.indra.jee.arq.spring.sofia2.script.api.Api;
public class ApiGroovy {
	public class ScriptException extends Throwable{
		public ScriptException(Exception e){
			super(e);
		}
		public ScriptException(Exception e, String message){
			super(message, e);
		}
		public ScriptException(String message){
			super(message);
```

```
		}
	}
	private Api api;
	public ApiGroovy(){
		api = new Api();
	}
	public String insert(String ontologyName, String ontology) throws ScriptException{
		try{
			return api.insert(ontologyName, ontology);
		}catch(Exception e){
			throw new ScriptException(e);
		}
	}
	public String rolback(String ontologyName, String id) throws ScriptException{
		try{
			return api.rolback(ontologyName, id);
		}catch(Exception e){
			throw new ScriptException(e);
		}
	}
	public String getAtribute(String ontology, String atribute) throws ScriptException{
		try{
			return api.getAtribute(ontology, atribute);
		}catch(Exception e){
			throw new ScriptException(e);
		}
	}
```

```
	public void sendMail(String to, String subject, String msg) throws
ScriptException{
		try{
			api.sendMail(to, subject, msg);
		}catch(Exception e){
			throw new ScriptException(e);
		}
	}
}
```

3.3 Variáveis da Plataforma.

Durante a execução, o motor de Scripting disponibiliza uma série de informação para que os scripts a possam utilizar:

`ontology`	Dados da Ontologia inserida ou atualizada.
`ontologyName`	Nome da Ontologia inserida ou atualizada.
`ontologyId`	Lista com a Id da instância da Ontologia inserida das instâncias atualizadas.
`typeMessage`	Tipo de operação que desencadeou o Script INSERT/UPDATE
`sessionKey`	SessionKey que realizou a inserção ou a atualização.

Estas variáveis estão acessíveis nos scripts mediante eventos e não nos scripts temporizados.

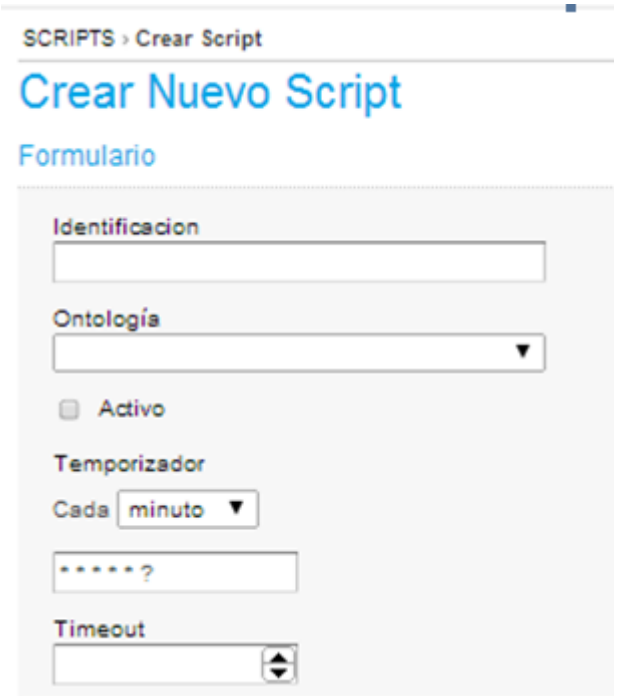
`scriptName`	Nome do Script lançado.

Esta variável está disponível em todos os Scripts.

`Error`	Exceção gerada.

Esta variável apenas tem valor nos Scripts do bloco Erro

3.4 Criação de um Script



Quando criamos um Script, devemos fornecer a informação seguinte:

- **Identificação** Valor único que identifica o nosso Script.
- **Ontologia** Ontologia que provocará a execução do Script.
- **Ativo** para ativar e desativar o Script
- **Temporizador** Indica com que frequência é necessário lançar o Script (Script Temporizado)
- **Timeout** Tempo máximo permitido ao script para a sua execução (se for superior ao tempo indicado pela plataforma, é considerado o da plataforma)

Em função dos parâmetros de entrada, um Script converte-se num:

- **Script Temporizado**: Valor temporizador selecionado
- **Script mediante evento**: Valor Ontologia selecionado)

Vejamos um exemplo de um Script que se executa antes da inserção ou atualização da Ontologia **SensorHumidade**.

- O **bloco If** 

(No Script associado à ontologia é obrigatório) define uma condição que devolve um valor *true* ou *false* e que provocará a execução do bloco ***then*** ou ***else***.

```
api = new ApiGroovy();
texto = api.getAtribute(ontology, "SensorHumedad.identificador");
if (texto=="ST-TA114"){
    return true;
}else{
    return false;
}
```

No exemplo anterior, a primeira coisa que fazemos é inicializar uma das operações que temos definidas.

```
api = new ApiGroovy();
```

A seguir, armazenamos numa variável o valor do atributo **SensorHumidade.identificador** da Ontologia inserida.

```
texto = api.getAtribute(ontology, "SensorHumedad.identificador");
```

Também é possível efetuar a avaliação previamente, se a operação for de inserção ou de atualização.

```
if (typeMessage=="INSERT"){
```

Por último e ao ter os valores sobre os quais pretendemos realizar a avaliação do bloco *if*, executamos a mesma.

```
if (texto=="ST-TA114"){
    return true;
}else{
    return false;
}
```

Em função da avaliação do bloco anterior, se existir (Nos Scripts Temporizados é opcional), executar-se-á o bloco *Then* ou *Else*.

- O **bloco *then*** executa-se quando a condição de ***If*** é ***true***

```
api = new ApiGroovy();
try{
    borrado = api.rolback(ontologyName, ontologyId[0]);
    creado = api.insert(ontologyName, ontology);
    texto = api.getAtribute(ontology, "SensorHumedad.identificador");
    c = new systemprinter();
    c.print("Borrado id "+borrado);
    c.print("Creado id "+creado);
    c.print("Valor del Atributo SensorHumedad.identificador "+texto);
    c.print("Operacion Lanzada "+typeMessage);
    api.sendMail("sofia2oncloud@gmail.com", "CASO THEN", ontology);
}catch(ScriptException e){
    api.sendMail("sofia2oncloud@gmail.com", "ERROR SCRIPT", e);
}

```

  - `borrado = api.rolback(ontologyName, ontologyId[0]);` Apagamos a instância da Ontologia que se inseriu.
  - `creado = api.insert(ontologyName, ontology);` Inserimos novamente a Ontologia que inserimos através da mensagem SSAP.
  - `c = new systemprinter();` Disponibilizamos outra biblioteca de operações.
  - Gerimos as exceções que podem ser geradas, assim ao produzir-se um erro enviamos um e-mail com o assunto “ERROR SCRIPT” assim como a exceção gerada.

```
}catch(ScriptException e){
    api.sendMail("sofia2oncloud@gmail.com", "ERROR SCRIPT", e);
}
```

- O **Bloco *ELSE*** é opcional e é executado quando não se cumpre a condição do Bloco *IF* e se foi definida alguma operação.

```
api = new ApiGroovy();
api.sendMail("sofia2oncloud@gmail.com", "CASO ELSE", ontology);
```

Neste caso, enviamos um e-mail com a indicação no assunto CASO ELSE e enviamos a Ontologia que desencadeou esta execução.

- O **Bloco *ERROR*** é opcional e executa-se quando se produz um erro não controlado em algum dos casos anteriores.

  Neste exemplo, envia-se um e-mail indicando que se produziu um erro e o erro produzido.

```
1 api = new ApiGroovy();
2 api.sendMail("sofia2oncloud@gmail.com", "ERROR SCRIPT", error);
```

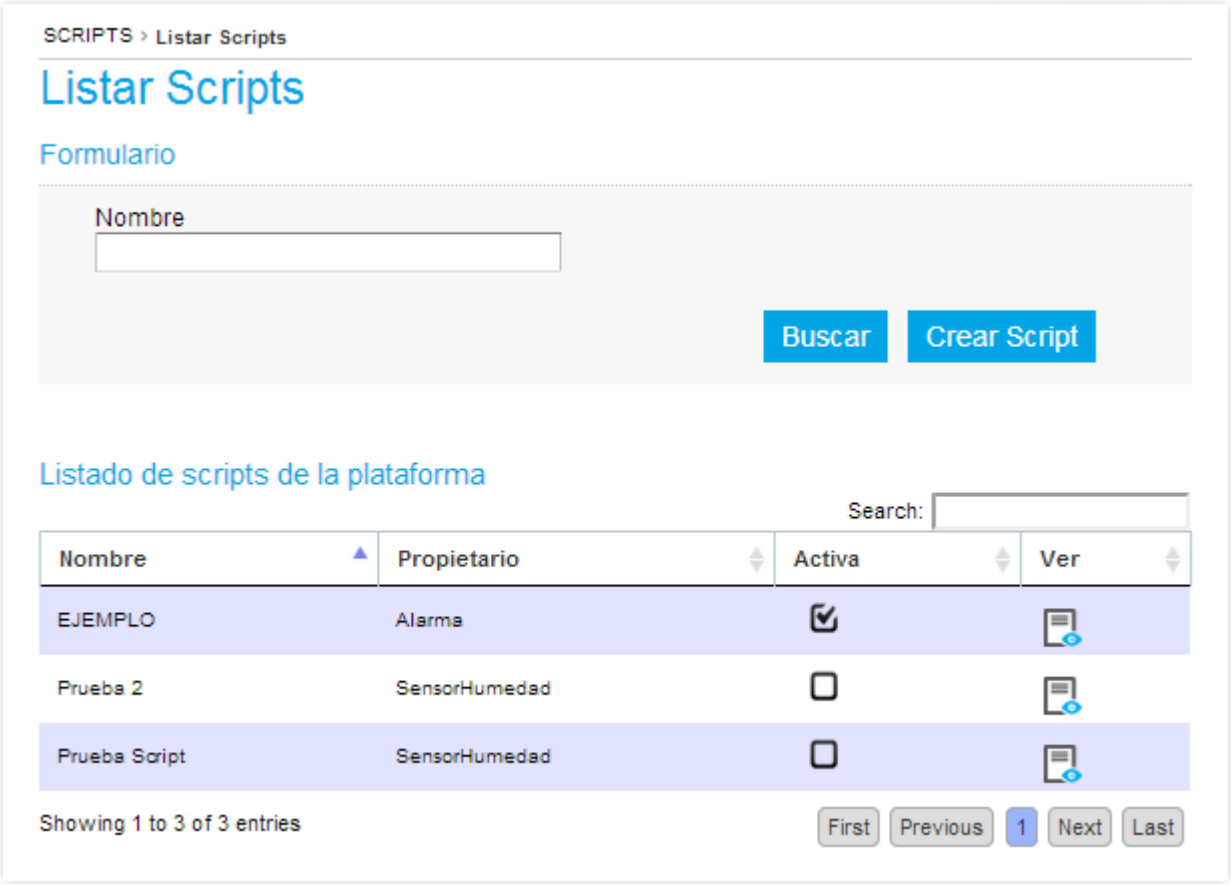
4 UI PARA CRIAÇÃO DE SCRIPTS PERSONALIZADOS

SOFIA2 permite a criação de scripts e de APIs (operações) a partir da Consola Web de Configuração e do respetivo API REST.

Em função do rol poderá:

- Gerir APIs: utilizador administrador
- Gerir Scripts: utilizador administrador e colaborador.

4.1 Lista de Scripts

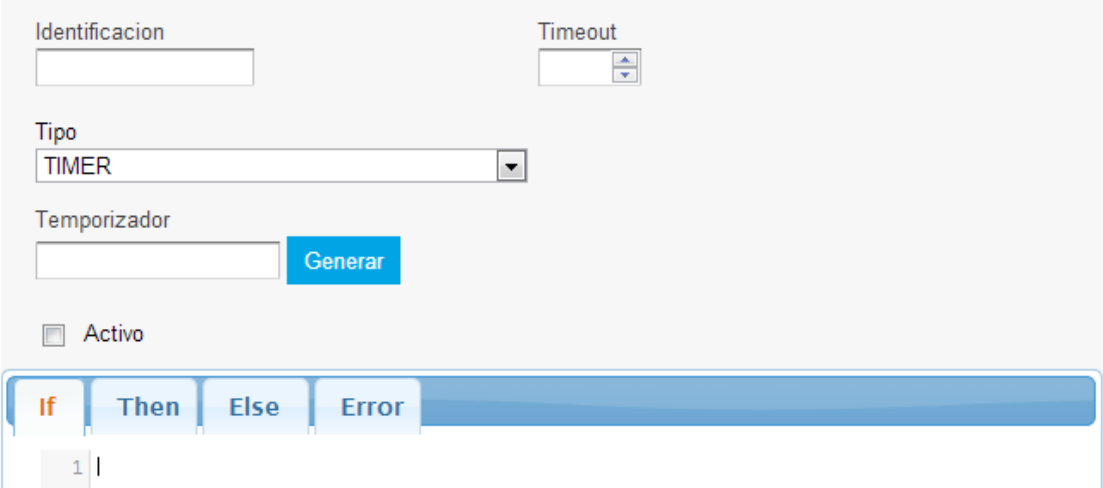


Dependendo do rol do utilizador terá acesso a uma parte:

- Os **Scripts temporizados** podem apenas ser criados pelos Administradores
- Os **Scripts associados a uma ontologia**:
    - Um administrador poderá efetuar uma gestão completa
    - Um colaborador apenas poderá gerir os seus próprios scripts, que poderão utilizar ontologias, sobre as quais o utilizador tiver permissões.

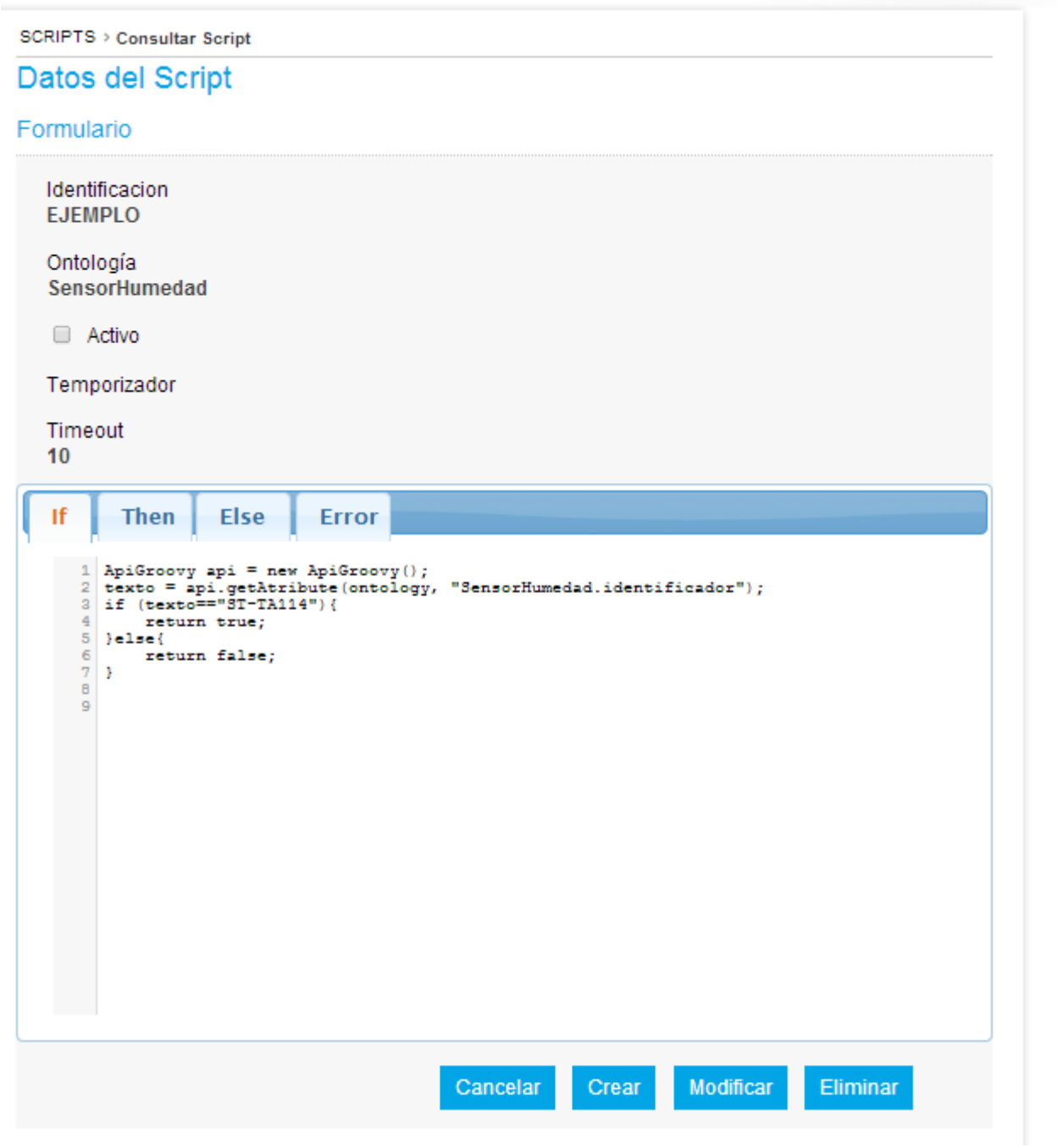
- Os **Scripts associados a uma ontologia principal**:
    - Um administrador poderá efetuar uma gestão completa.
    - Um colaborador apenas poderá gerir os seus próprios scripts, que poderão utilizar ontologias, sobre as quais o utilizador tiver permissões
- Os **Scripts do tipo Cep:** podem ser criados pelos utilizadores com o rol Administrador e Colaborador.

4.2 Script Temporizados



- Os Scripts de tipo temporizado não têm nenhuma Ontologia associada.
- Definem o tipo de temporização através de elementos finais e por fim esta temporização converte-se numa temporização de tipo CRON.
- Este tipo de Script tem de ter obrigatoriamente definido o bloco ***Then*.**
- De forma opcional poderá ter definido um bloco ***If***, ***Else*** e ***Error***.
- Define-se um bloco ***If*** sempre que é lançado o Script. Em função do valor indicado no atributo Temporizador avalia-se a condição e se esta for cumprida executa-se o bloco *Then* ou *Else*. Se não existir condição, executa-se sempre o bloco *Then*.

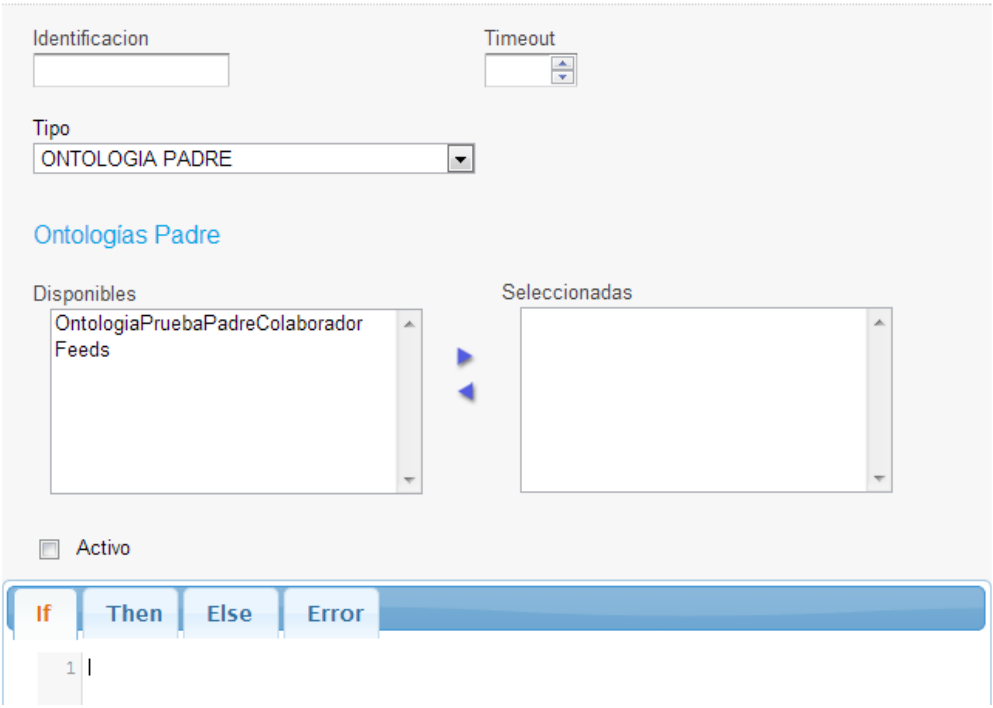
4.3 Script associados a Ontologias



- Os Scripts associados a uma Ontologia são lançados quando se realiza uma inserção ou atualização da Ontologia associada.
- Este tipo de Script tem de ter obrigatoriamente definido o bloco ***If*** e ***Then***
- Opcionalmente, poderá ter definido um bloco ***Else*** e ***Error***.
- Sempre que se realizar uma inserção ou atualização de uma **Ontologia,** lançam-se os Scripts associados a esta. O primeiro passo é executar o bloco ***If***, que devolverá um

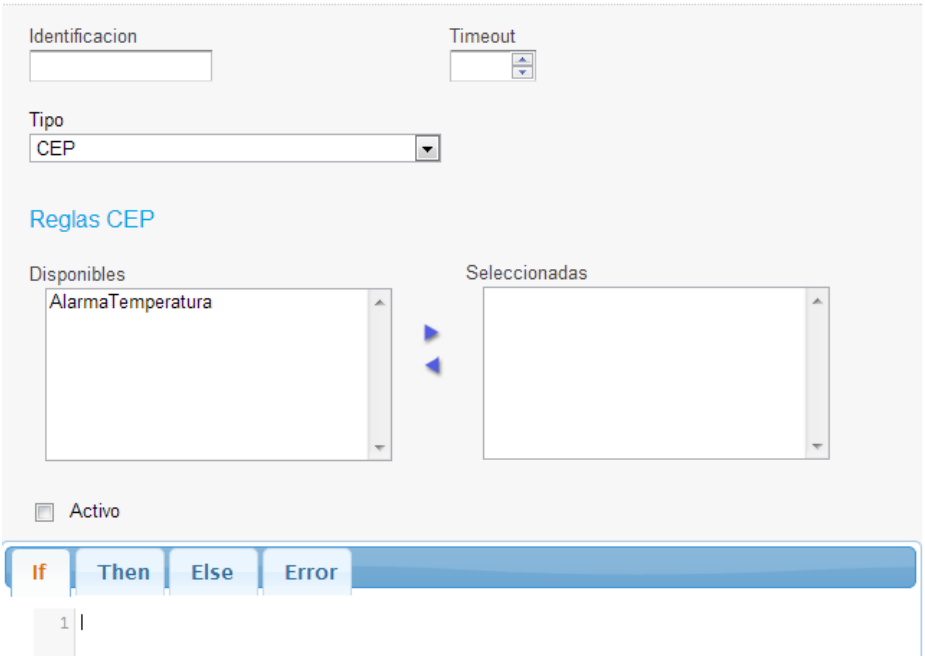
valor *true* ou *false.* Se se cumprir a condição, executa-se o bloco *Then*, e se não se cumprir, está definido o bloco *Else*.

4.4 Script associados a Ontologias Principais



- Os Scripts associados a uma Ontologia Principal são lançados quando se realiza uma inserção ou uma atualização da Ontologia associada.
- Este tipo de Script tem de ter obrigatoriamente definido o bloco ***If*** e ***Then***
- Opcionalmente, poderão ter um bloque ***Else*** e ***Error*** definidos.
- Sempre que se realiza uma inserção ou atualização de uma **Ontologia Principal,** lançam-se os Scripts associados a esta, o primeiro passo é executar o bloco ***If***, que tem de devolver um valor *true* ou *false.* Se se cumprir a condição, executa-se o bloco *Then*, e se não se cumprir está definido o bloco *Else*.

4.5 Script de Tipo Cep



- Os Scripts de tipo Cep têm a principal característica de que se executam sempre que se produz um evento indicado pelo criador do Script
- Este tipo de Script tem de ter obrigatoriamente definido o bloco ***If*** e ***Then***
- Opcionalmente, poderão ter um bloco ***Else*** e ***Error*** definidos.
- Cada vez que se executar o script, avaliar-se-á a condição que se indica no campo *IF*, devolvendo o controlo ao contentor de Script, que por sua vez devolverá um valor *true* ou *false*. Se se cumprir a condição, executa-se o bloco *Then*, e se não se cumprir está definido o bloco *Else*.

5 SEGURANÇA NA EXECUÇÃO DE SCRIPTS

O módulo de Script da plataforma SOFIA2 dispõe de 2 medidas de segurança com a finalidade de evitar que a execução de um Script suponha um risco para a integridade do sistema.

5.1 Segurança Ativa.

Baseada em Java Policy (mais informação sobre esta especificação nos seguintes links **http://docs.oracle.com/javase/6/docs/technotes/guides/security/permissions.html** e **http://docs.oracle.com/javase/6/docs/technotes/guides/security/PolicyFiles.html**)

Permite definir as operações que a JVM suportará, sendo esta a que impedirá que se execute o código que não cumpra as restrições de segurança.

Este nível de segurança é aplicado aos Script assim como às operações definidas.

A definição de um TimeOut a nível da plataforma garante que nenhum Script esteja em execução, para além do tempo máximo permitido, o que fará com que o *Thread,* no qual este é lançado, seja eliminado quando ultrapassar o tempo máximo de execução.

5.2 Segurança Passiva.

Com base num analisador sintático, impede a utilização de estruturas não permitidas quando definimos um Script.

Os Script têm as seguintes restrições.

- Definir Classes.
- Definir Métodos.
- Importar Classes.
- Utilizar o *System*.
- Utilizar as classes:
  BufferedInputStream, BufferedOutputStream, BufferedReader, BufferedWriter, ByteArrayInputStream, ByteArrayOutputStream, CharArrayReader, CharArrayWriter, Console, DataInputStream, DataOutputStream, File, FileDescriptor, FileInputStream, FileOutputStream, FilePermission, FileReader, FileWriter, FilterInputStream, FilterOutputStream, FilterReader, FilterWriter, InputStream, InputStreamReader, LineNumberInputStream, LineNumberReader, ObjectInputStream, ObjectInputStream.GetField, ObjectOutputStream, ObjectOutputStream.PutField, ObjectStreamClass, ObjectStreamField, OutputStream, OutputStreamWriter, PipedInputStream, PipedOutputStream, PipedReader, PipedWriter, PrintStream, PrintWriter, PushbackInputStream, PushbackReader, RandomAccessFile, Reader, SequenceInputStream, SerializablePermission, StreamTokenizer, StringBufferInputStream, StringReader, StringWriter, Writer, Class, ClassLoader, Compiler, InheritableThreadLocal, Process, ProcessBuilder,ProcessBuilder.Redirect, Runtime, RuntimePermission, SecurityManager, StackTraceElement, Thread, ThreadGroup, ThreadLocal, Authenticator, CacheRequest, CacheResponse, ContentHandler, CookieHandler, CookieManager, DatagramPacket, DatagramSocket, DatagramSocketImpl, HttpCookie, HttpURLConnection, IDN,

Inet4Address, Inet6Address, InetAddress, InetSocketAddress, InterfaceAddress, JarURLConnection, MulticastSocket, NetPermission, NetworkInterface, PasswordAuthentication, Proxy, ProxySelector, ResponseCache, SecureCacheResponse, ServerSocket, Socket, SocketAddress, SocketImpl, SocketPermission, StandardSocketOptions, URI, URL, URLClassLoader, URLConnection, URLDecoder, URLEncoder, URLStreamHandler.

Para efetuar qualquer tipo de ação fora dos blocos de controlo, devem invocar-se as operações definidas que não têm estas restrições.

Quando criamos um Script através do UI, verificamos que este não viola estas validações e que pode ser armazenado na base de dados para a sua execução.