

A.1 Service Broker

Role(s): Customer

Component(s): Semantic Enhanced Service Selector (SESS)
SLA Translator
Service Discovery Registry (SDR)

License: LGPL

A.1.1 Installation

A.1.1.1. Installation Requirements

- SESS and SLA Translator: Java 5 or higher, Tomcat 5.5 or higher;
- Service Discovery Registry: Windows Platform, Microsoft .NET Framework 3.5 or higher, IIS with WCF capability enabled, Java 5 or higher, Tomcat 5.5 or higher.

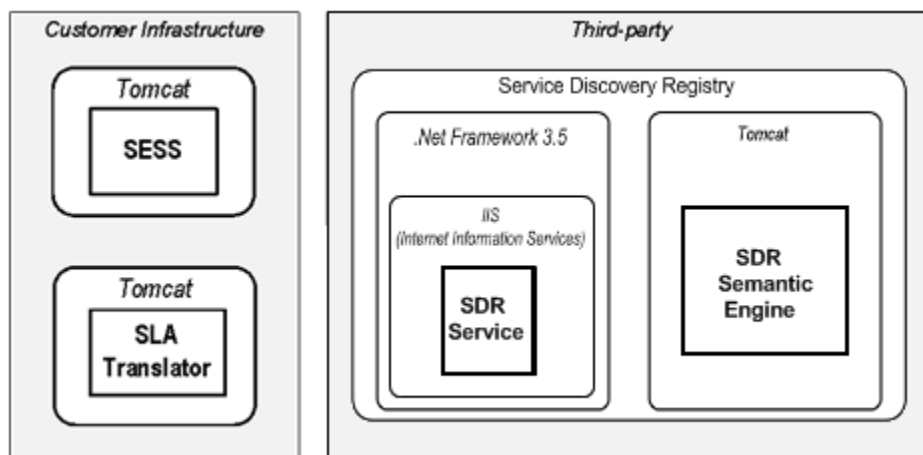


Figure 4: Service Broker Toolkit components deployment

A.1.1.2. Deployment Tips

The SESS and the SLA Translator, as Web services, can be installed in different hosts and interact through their Web service interface. However, in general, it is recommended to install both in the same host. The Service Discovery Registry is generally installed in a third-party organisation, which provides discovery services to customers. The Service Discovery Registry can also be deployed in more than one instance in different hosts. This way it is possible to achieve a distributed network of service registries, and the discovery of services is improved since performed through this network (see Figure 5).

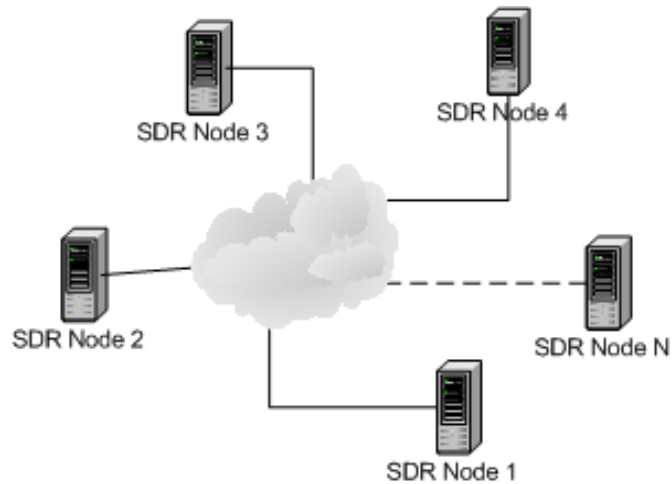


Figure 5: Service Discovery Registry Network

A.1.2 Software Installation

The Service Broker toolkit installer installs all the application / components required to provide the functionalities required to select the best service which fits the customer requests from providers' offers. The installation of these components is described in the following.

Semantically Enhanced Service Selector

Prerequisites

The Semantically Enhanced Service Selector is a Web Service developed in Java, so that it can be deployed in any Operating System. It requires Java 5 or later and Apache Tomcat 5.5 to be installed. In case Tomcat is not installed in the computer, the installation process is aborted.

Installation

The BREIN installer provides a wizard so that the customer can specify some configuration information for the installation.



Figure 6: Service Selector Setup Wizard

After all the steps are done (like accept the license terms), the application is installed and has some desktop links to access to the component functionality like Figure 7 shows.

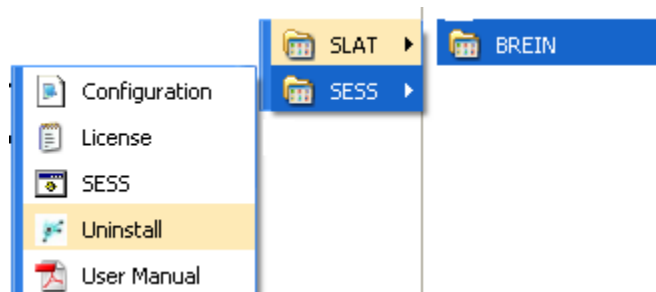


Figure 7: Windows Start Menu Shortcuts

The SESS component is deployed as a Web service in a `.war` file. By using the BREIN installer, the `.war` file is just deployed inside the tomcat server container. In order to run the application, the user can just link the SESS file which start tomcat.

To configure the component, the user can update the configuration file (`sess.properties`) with the following properties:

- `log_file`: the path of the file where the logs are stored.
- `log_level`: level of the log (FINE, CONFIG, INFO, WARNING, SEVERE)
- `url_slat`: the URL where the SLAT service is deployed

Finally, the services should be accessible locally at the following URL:

<http://localhost:8080/SESS-2.0/SESSService>

SLA Translator

Prerequisites

The SLA translator is a Web service developed in Java, so that it can be deployed in any Operating System. It requires Java 5 or later and Apache Tomcat 5.5 to be installed. In case Tomcat is not installed in the computer, the installation process is aborted.

Installation

The BREIN installer provides a wizard so that the customer can specify some configuration information for the installation.



Figure 8: SLA Translator Setup Wizard

After all the steps are done (like accept the license terms), the application is installed and has some desktop links to access to the component functionality like Figure 9 shows.

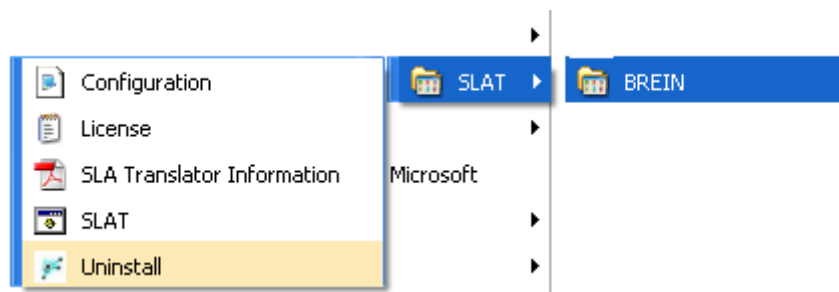


Figure 9: Windows Start Menu Shortcuts

The SLAT component is deployed as a Web service in a `.war` file. By using the BREIN installer, the `.war` file is just deployed inside the tomcat server container.

In order to run the application, the user can just link the SLAT file which start tomcat.

To configure the component, the user can update the configuration file (`slat.properties`) with the following properties:

- `log_file`: the path of the file where the logs are stored.
- `log_level`: level of the log (`FINE`, `CONFIG`, `INFO`, `WARNING`, `SEVERE`)
- `qos`: file where the QoS ontology is located
- `namespaceqos`: the namespace of the ontology
- `thesaurus`: file where the thesaurus for SKOS is located.

Finally, the services should be accessible locally at the following URL:

<http://localhost:8080/SLAT-2.0/SLATService>

Service Discovery Registry

Prerequisites

The Service Discovery Registry component requires Microsoft .NET Framework 3.5 or higher, IIS with WCF capability enabled, Java 5 or higher, Tomcat 5.5 or higher.

Installation

The Service Discovery Registry is shipped with three sub-components: SDRService, SDRWeb and SDRSemanticEngine. The SDRService component must be deployed on Microsoft Internet Information Services (IIS) on Windows XP Professional, Windows Vista, Windows Server 2003 and Windows Server 2008. The SDRSemanticEngine must be deployed on Tomcat Server.

In order to deploy the complete SDR package the user has to execute the related installer following the instruction for a correct configuration. The installer by default will create a 'virtual' directory in the IIS called "SDRService". By opening the IIS Management Console the user will have to make these sub-directories application directories (See Figure 10).

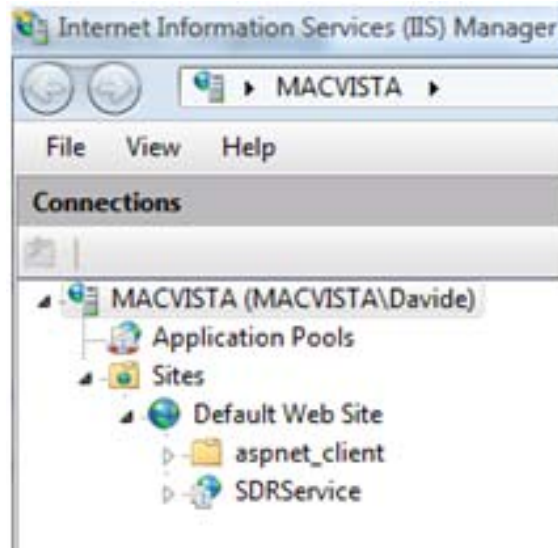


Figure 10: SDR Services in the IIS

The services should be accessible locally at the following URLs:

- <http://localhost/SDRService/SDService.svc>
- <http://localhost/SDRWeb/Default.aspx>

The execution of the installer creates the following configuration files (SDRService\Web.config and SDRWeb\Web.config), in the case the user wants to change them manually, see below:

- Update SDRSemanticEngine and SATSLARepositoryService endpoints

```
<client>
  <endpoint address="http://localhost:8080/SDRSemanticEngine/services/MatchmakerInterface"
    binding="basicHttpBinding" bindingConfiguration="MatchmakerInterfaceSoapBinding"
    contract="SemanticEngine.MatchmakerInterface" name="MatchmakerInterface" />

  <endpoint address="http://xxx.xxx.xxx.xxx/SATSLARepository/SATSLARepositoryService.asmx"
    binding="basicHttpBinding" bindingConfiguration="SATSLARepositoryServiceSoap"
    contract="SATSLARepositoryService.SATSLARepositoryServiceSoap"
    name="SATSLARepositoryServiceSoap" />
</client>
```

- Update the application settings:
 - BreinSdWcfService_WebRef_SATSLARepository_SATSLARepositoryService, endpoint of the SATSLARepositoryService
 - BreinSdWcfService_WebRef_MatchmakerInterface_MatchmakerInterfaceService, endpoint of the SDRSemanticEngine
 - SemXConeLocalAddress, defines the local node address which is used to communicate with the others nodes of the network
 - SemXConeLocalPort, defines the local node port which is used to communicate with the others nodes of the network
 - SemXConeIsBootstrapNode, defines if the local SD node is a bootstrap node of Service Discovery Registries Network
 - SemXConeHostAddress, defines the SD node address to start the bootstrap process
 - SemXConeHostPort, defines the SD node port to start the bootstrap process

- SemanticEngineHostAddress, node address of the SDRSemanticEngine
- SemanticEngineHostPort, node port of the SDRSemanticEngine
- CatOntology, the URI to category ontology which classify the published services

```

<applicationSettings>
  <setting name="BreinSdWcfService_WebRef_SATSLARepository_SATSLARepositoryService"
    serializeAs="String">
    <value>http://xxx.xxx.xxx.xxx/SATSLARepository/SATSLARepositoryService.asmx</value>
  </setting>
  <setting name="BreinSdWcfService_WebRef_MatchmakerInterface_MatchmakerInterfaceService"
    serializeAs="String">
    <value>http://localhost:8080/SDRSemanticEngine/services/MatchmakerInterface</value>
  </setting>
  <setting name="SemXConeLocalAddress" serializeAs="String">
    <value>xxx.xxx.xxx.xxx</value>
  </setting>
  <setting name="SemXConeLocalPort" serializeAs="String">
    <value>4444</value>
  </setting>
  <setting name="SemXConeIsBootStrapNode" serializeAs="String">
    <value>True</value>
  </setting>
  <setting name="SemanticEngineHostAddress" serializeAs="String">
    <value>127.0.0.1</value>
  </setting>
  <setting name="SemXConeHostAddress" serializeAs="String">
    <value>xxx.xxx.xxx.xxx</value>
  </setting>
  <setting name="SemXConeHostPort" serializeAs="String">
    <value>4445</value>
  </setting>
  <setting name="SemanticEngineHostPort" serializeAs="String">
    <value>8080</value>
  </setting>
  <setting name="CatOntology" serializeAs="String">
    <value>http://localhost:8080/webdav/ontologies/UNSPSC.owl</value>
  </setting>
</applicationSettings>

```

After the installation the user has to deploy the SDRSemanticEngine into the webapps directory of the local instance of Tomcat application server. The installer modifies your windows firewall configuration as well by adding the follow rule:

- Accept any TCP communication (in/out) from SemXConeLocalPort

Note: At the end of the installation process, execute "iisreset" from command line to restart IIS. This is needed to synchronize IIS with the new system environment variables.

A.1.3 Usage Instructions

The Service Broker toolkit provides the functionalities required to select the best service which fits the customer requests from providers' offers. Basically, this selection involves both the discovery and selection of the service considering both semantic functional and non-functional properties. The non-functional properties are described in terms of SLA terms such as Quality of Service (QoS) metrics located in SLA templates. All the descriptions will be semantically enhanced by the usage of OWL-S for service functional descriptions and semantic annotations of SLA files for the non-functional descriptions.

The broker thereby can return the 'best fit', which may however not meet all conditions specified by the customer (the Calling Client in Figure 11). This functionality will leverage the service discovery by means of the selection of the

best choice of the discovered candidate services that are available to perform the service requested by the customer.

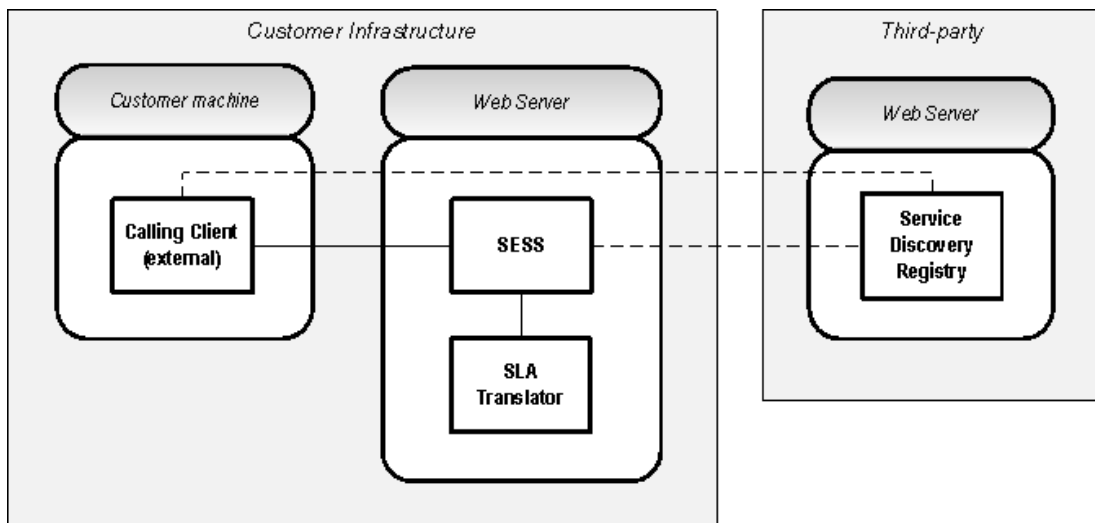


Figure 11: Service Broker Toolkit Components

As seen in the previous section, the Service Broker toolkit is composed of three components: the Semantic Enhanced Service Selector (SESS), the SLA Translator, and the Service Discovery Registry (SDR), see Figure 11.

The toolkit installs all the components as Web services. The SESS performs the service selection, contacting the SLA Translator to translate SLA terms if required. The selection is made from a list of available services and depends on the SLA templates content. The Service Discovery Registry is used for service publication and advertising; to request for the list of published services, to remove a service publication, and so forth.

A general interaction between the components is as follows:

- The Client (in the BREIN framework the Workflow Enactor) sends to the Service Discovery Registry the service profile representing the service to be searched;
- The Service Discovery Registry matches the received profile against the profiles stored in the registry;
- The Service Discovery Registry sends to the Client the matched profiles together with the associated SLA templates;
- The Client sends the services profiles and the associated SLA templates to the SESS;
- The SESS translates SLA terms through the SLA Translator, if required;
- The SESS selects the proper service and sends it to the Client.

It is worth noticing that the SESS, alternatively, can directly interact with the Service Discovery Registry component (in Figure 11 the dashed lines denote alternative required interactions). So, two behaviours are possible. In the first case, the SESS receives the list of available services from the calling client and makes the selection. In the second case, the SESS receives the services list directly from the Service Discovery Registry and makes the selection. The first approach is the standard approach and it is used in the Virtual Engineering

scenario where the Workflow Enactor contacts the Service Discovery Registry to get the services list and then the SESS to perform the service selection.

SLA Translator

Since BREIN tries to provide simplicity to final users, providers are able to define their metrics in a language, or in a way they want. As BREIN offers a non-functional properties selection based on QoS metrics located in SLA templates, some semantic support is required for the SLA templates file.

SLA translator tries to aim the provision to every SLA aware component of a way to have a common terminology, although service and resource providers have defined their own SLA metrics. In this matter, those components will be able to understand the new metrics defined by providers. The idea is simple. Each time a SLA component cannot process a metrics, it asks to SLA translator for a predefined one.

Table 3 - SLAT Interface

Method	<code>String strQoSmetric obtainPredefinedMetric (URI metricConcept)</code>
Arguments	URI <i>metricConcept</i> – A set of URI of the SLA templates candidates
Return Value[s]	String <i>QoSmetric</i> - The strQoSmetric predefined and understandable by SLA components
Description	This method returns the predefined metric to be used by SLA components from new metrics defined by providers
Method	<code>URI[] concepQoSmetrics obtainQoSmetricsInLanguage(String language)</code>
Arguments	String <i>language</i> – The language to obtain metrics
Return Value[s]	URI[] <i>concepQoSmetrics</i> – The QoS metrics to annotate SLA templates
Description	This method obtain all the QoS metrics to be used by SATSLA GUI

Service Discovery Registry

The Service Discovery Registry (SDR) functionality is as follows:

- Allows the Service Providers to publish their services into any of Service Discovery Registry from the Network;
- Allows the Customers to find services they need by querying from any registry node of the network.

The SDR allows for matching customer's required profiles with the provider offers. Each service (and query, too) handled by the component is described using the OWL-S specification (Service Profile). Furthermore, the Service Provider specifies the signature of the advertised service through the ontology class of inputs, outputs and service's classification.

When a new service is published into the SDR, the SDR stores a local copy of service and update the index information used from the others peers of Service Discovery Registries Network. When a query is submitted, through an OWL-S Service Profile, the SDR uses a semantic engine to retrieve the local published services with one of the following degree of match: Exact Match, Plug-in Match, Subsumes match, Subsumed-by match, and Nearest-neighbour match. The first three degree of match is logic based and the last two are hybrid due to the required additional computation of syntactic similarity values.

If the size of collected results is not enough, the SDR would forward the query to others semantic relevant instance of SDR.

Table 4 - SDR Interface

Method	<code>void AddService(string <i>serviceId</i>, string <i>profile</i>, KVPair[] <i>additionalInfo</i>)</code>
Arguments	<code>string <i>serviceId</i></code> – Unique identifier of the service profile; it can be any string.
	<code>string <i>profile</i></code> – The service profile to be added, an XML document compliant with OWL-S specification.
	<code>KVPair[] <i>additionalInfo</i></code> – A list of generic info (key-value pairs) related to the service.
Return Value[s]	None
Description	Insert, or update if already exists, a service profile into the service discovery registry.
Method	<code>Void RemoveService(string <i>serviceId</i>)</code>
Arguments	<code>string <i>serviceId</i></code> - Unique identifier of the service profile; it can be any string.
Return Value[s]	None
Description	Remove a service profile from the service discovery registry.
Method	<code>String[] ListServices()</code>
Arguments	None
Return Value[s]	An array of string; each string is the identifier of a service profile.
Description	List all service profile identifiers contained into local node of the service discovery registry.
Method	<code>MatchedService[] MatchRequest(string <i>profile</i>)</code>
Arguments	<code>string <i>profile</i></code> - The service profile to be search, an XML document compliant with OWL-S specification. It can contains both semantic and SLA characteristics description.
Return Value[s]	See MatchRequestCustom operation.
Description	It executes the same search of the MatchRequestCustom operation, but with default parameters configure with setter operations.
Method	<code>MatchedService[] MatchRequestCustom(string <i>profile</i>, int <i>minDegreeOfMatch</i>, double <i>simThreshold</i>)</code>
Arguments	<code>string <i>profile</i></code> - The service profile to be search, an XML document compliant with OWL-S specification. It can contains both semantic and SLA characteristics description.
	<code>int <i>minDegreeOfMatch</i></code> - minimum degree of match parameter for the semantic search.
	<code>double <i>simThreshold</i></code> - similarity threshold parameter for the semantic search.
Return Value[s]	An array of MatchedService elements, sorted by semantic and SLA degree of match; each element contains the serviceld of the profile and the result parameters of the match; also the SLAs documents retrieved from the SATSLARepository and the service additionalInfo are returned.
Description	It performs the search of service profiles matching against the service profile request.
Method	<code>KVPair[] GetConfig()</code>
Arguments	None
Return Value[s]	KVPair[] – The list of Semantic Engine configuration parameters.
Description	Getter operation for the whole configuration of Semantic Engine.
Method	<code>string GetConfigParam(string <i>name</i>)</code>
Arguments	<code>string <i>name</i></code> - The name of the Semantic Engine configuration parameter to be get.

Return Value[s]	String - The value of configuration parameter required.
Description	Getter operation to retrieve from the local Service Discovery Registry node the Semantic Engine configuration.
Method	void getConfig (KVPair[] config)
Arguments	KVPair[] config - The list of Semantic Engine configuration parameters.
Return Value[s]	None
Description	Setter operation for the whole configuration of Semantic Engine.
Method	void setConfigParam (string name, string value)
Arguments	string name - The name of the Semantic Engine configuration parameter to be set. string value - The value of the Semantic Engine configuration parameter to be set.
Return Value[s]	None
Description	Set operation to update a local Semantic Engine configuration.
Method	void delConfigParam (string name);
Arguments	string name - The name of the configuration parameter to be delete.
Return Value[s]	None
Description	Delete a local Semantic Engine configuration.
Method	void initialize ()
Arguments	None
Return Value[s]	None
Description	Initialize manually the discovery engine, starting and configuring the backend, and loading the service registry from content of the persistent repository. This Operation is automatically executes at the first (any) operation invocation.
Method	void startupBackend ()
Arguments	None
Return Value[s]	None
Description	Start the backend container and join into Service Discovery Registries Network.
Method	void shutdownBackend ()
Arguments	None
Return Value[s]	None
Description	Stop the backend container and leave the Service Discovery Registries Network.
Method	void joinSDRNetwork ()
Arguments	None
Return Value[s]	None
Description	Join into Service Discovery Registries Network.
Method	void leaveSDRNetwork ()
Arguments	None
Return Value[s]	None
Description	Leave the Service Discovery Registries Network.
Method	SemanticEngineStatus getSemanticEngineStatus ()
Arguments	None
Return Value[s]	Useful information about the Semantic Engine Status.
Description	Get the status of the semantic engine.
Method	SemanticDistributedIndexStatus getSemanticDistributedIndexStatus ()
Arguments	None
Return	Useful information about the Semantic Distributed Index Status.

Value[s]	
Description	Get the status of the semantic distributed index.
Method	string GetSDIndexConfigParam (<i>string name</i>)
Arguments	<i>string name</i> - The name of the Semantic Distributed Index configuration parameter to be get.
Return Value[s]	<i>string value</i> - The value of the Semantic Distributed Index configuration parameter.
Description	Getter operation to retrieve from the local Service Discovery Registry node the Semantic Distributed Index configuration.
Method	void SetSDIndexConfigParam (<i>string name, string value</i>)
Arguments	<i>string name</i> - The name of the Semantic Distributed Index configuration parameter.
	<i>string value</i> - The value of the Semantic Distributed Index configuration parameter to be set.
Return Value[s]	None
Description	Set operation to update at the local Service Discovery Registry node the Semantic Distributed Index configuration.
Method	void DelSDIndexConfigParam (<i>string name</i>)
Arguments	<i>string name</i> - The name of the Semantic Distributed Index configuration parameter to be delete.
Return Value[s]	None
Description	Delete from the local Service Discovery Registry node the Semantic Distributed Index configuration.
Method	KVPair[] GetSDIndexConfig ()
Arguments	None
Return Value[s]	KVPair[] <i>config</i> - The list of Semantic Distributed Index configuration parameters.
Description	Getter operation for the whole configuration of Semantic Distributed Index.