

## B.1 Workflow Framework

**Role(s):** Service Provider, Customer

**Component(s):** Workflow Designer  
Workflow Enactor  
Workflow Representation

**Toolkit(s):** Service Broker  
Semantic Enhanced Service Polishing and Registry

**License:** GPL V3

### B.1.1 Installation

#### B.1.1.1. Installation Requirements

- Workflow Designer, Workflow Representation, and Workflow Enactor: Java 6 or higher;
- 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;
- SATSLA Repository: Microsoft .NET Framework 3.5 and on Microsoft SQL Server;
- SATSLA GUI: Microsoft .NET Framework 3.5.

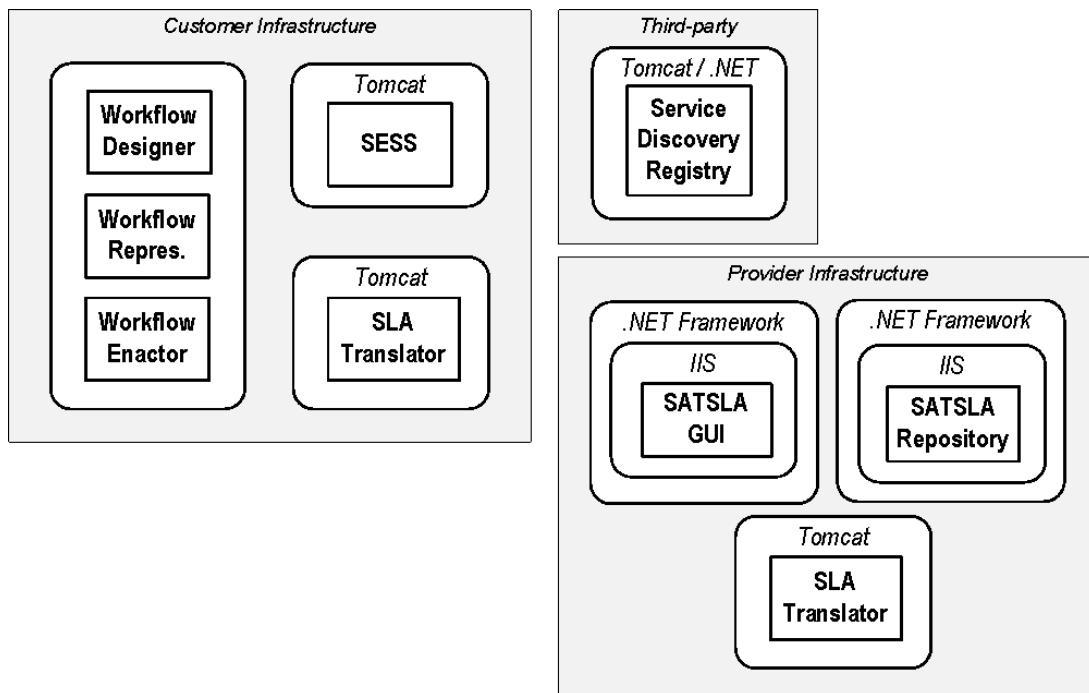


Figure 45: Workflow Management Framework components deployment

### B.1.1.2. Deployment Tips

The Workflow Designer, the Workflow Representation and the Workflow Enactor are application and libraries that interact closely. For this reason they are supposed to be installed in the same host. The SESS and the SLA Translator, as Web services, can be installed in different hosts. In general, it is recommended to install both the SESS and the SLA Translator in the same host as the one for the Workflow Designer, Workflow Representation and Workflow Enactor. SATSLA GUI, SATSLA Repository, and the SLA Translator (service provider instance) are generally installed in the same host. Normally this is a host provided by the Service Provider. In practice, the Service Provider uses these components to edit SLA templates, representing contracts, for each of the services they want to expose. SLA Templates can contain sensible terms that may be better to maintain stored in an inter-organization component. A different approach applies for the Service Discovery Registry. This components being a registry is generally installed in a third-party organisation providing 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 46).

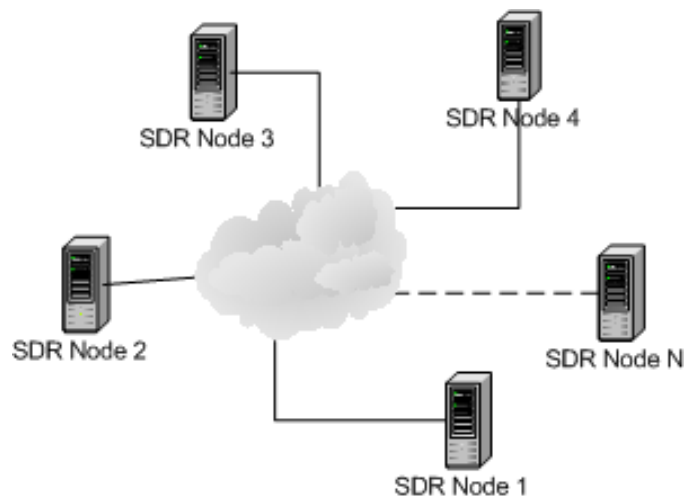


Figure 46: Service Discovery Registry Network

Service invocations through the Workflow Enactor are better if performed adding communication security (in- and out-going messages). This can be achieved installing an instance of the Gateway component in the Customer Infrastructure. This also grants virtualization of the invoked services endpoint references (EPRs).

### B.1.2 Software Installation

The Workflow Management Framework installer installs all the applications / components required for the modelling, enactment and concretization of abstract workflows. The installer firstly installs the Service Broker toolkit, then the Service Publishing toolkit and finally the Workflow Designer, the Workflow Enactor and the Service Discovery Registry. For more details on the Service Broker and the Service Publishing toolkits refers to the specific toolkits installation manuals. Installation of the Workflow Designer, the Workflow Representation, and the Workflow Enactor is described in the following sections.

The list of installers:

- Workflow Framework: SetupVIM&SWPT.exe, this installer will install all the individual components of the framework. The toolkits must be installed separately with their own installers, for more information refer to the toolkit installation manuals.
- Workflow Designer: SetupSWPT.exe This installer will individually install the designer.
- Workflow Enactor: SetupVIM.exe This installer will individually install the Enactor.

The last two installer are for individual component installation, this has been done for cases where the whole framework is not needed.

## Workflow Designer

### Prerequisites

The Workflow Designer is a Java application having no particular installation prerequisites apart from a set of libraries. These libraries are all provided with the installation package:

- JGraph;
- L2fprod Common;
- OWL-WS API (Mindswap OWL-S API extended version);
- Jena;
- Pellet.

A Java runtime environment is also required (Java 1.6 or greater).

### Installation

The installation is quite straightforward. Once the package is unpacked (see Figure 47), the application can be launched through the `SWPT.bat` batch file (issuing the command `SWTP`) or directly from the Windows Start Menu.



Figure 47: Workflow Designer (SWPT) Installer

## Workflow Representation

### Prerequisites

The Workflow Representation is a Java set of libraries having no particular installation prerequisites apart from Jena and Pellet (provided with the installation package).

### Installation

The component installation is performed during the whole package installation. The component makes available a set of libraries used by both the Workflow Designer and the Workflow Enactor. No component configuration is required.

## Workflow Enactor

### Prerequisites

The Workflow Enactor is Operating System independent as it is based on Java. It has been tested mainly in Linux, OSX, and Windows XP. It requires Java Version 1.5 or later.

All the java packages that the Workflow Enactor depends on are located in Maven public repositories or in the public repository at IT Innovation.

Shell scripts have been developed mainly for Unix bash shell, but some are also provided for Windows XP command shell.

### Installation

The Workflow Enactor consists of the following components (some of them are plug-in components to communicate with external services that are part of the BREIN framework):

- the `adaptive-api` component is the core process object model of the enactor;
- the `enactor` component contains the evaluation and execution environment;
- the `cxfutils` component contains utilities useful for invoking BREIN services through the BREIN Gateway;
- The `axis-utils` contains utilities useful for invoking WSDL services;
- The `simpleCLI` provide a simple Command Line Interface (CLI) mainly for testing purposes but it supports all available functionality of the enactor;
- Evaluation Plug-ins can be optionally installed. They implement one of the `ProcessDiscoverer`, `ProcessSelector`, `ProcessNegotiator` interfaces of the enactor:
  - The `ServiceDiscoveryRegistryPlugin` (`ProcessDiscoverer`) provides functionality required to contact the Service Discovery Registry (SDR) component while searching for services;
  - The `SESSPlugin` (`ProcessSelector`) provides functionality to access the Semantically Enhanced Service Selector (SESS) to select a service among the available ones returned by the Service Discovery Registry;

- The `SLANegotiatorPlugin (ProcessNegotiator)` is required to contact the SLA Negotiator component in order to negotiate the SLA terms (QoS) of the service being used.

The Workflow Enactor can be installed without the plug-ins. Nevertheless, their installation is generally required, to allow the components taking part in the framework, to properly provide workflow enactment functionality.

The installer automatically installs all the required components, giving the user the possibility to choose if to install the Command Line Interface and the plug-ins (see Figure 48).

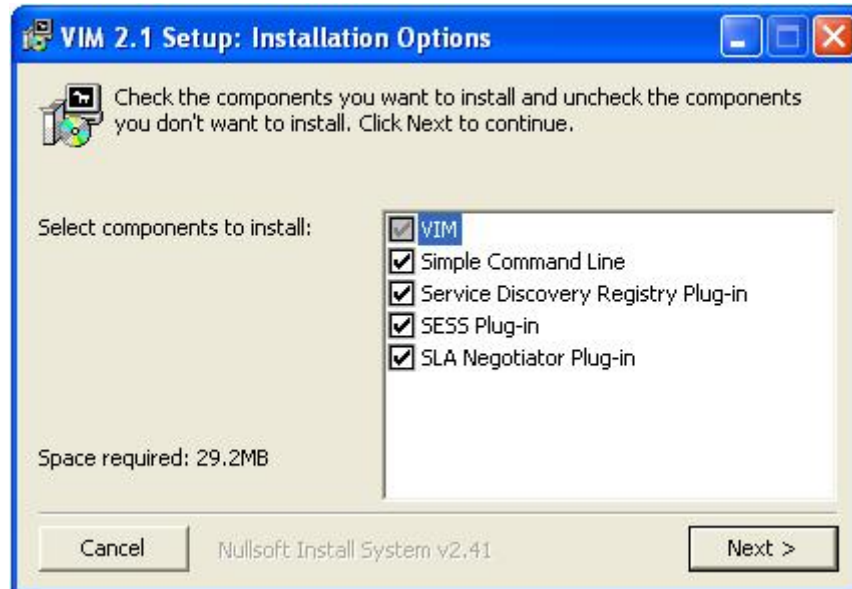


Figure 48: Workflow Enactor Installer

## Configuration

Considering `ENACTOR_HOME` being the Workflow Enactor installation folder, the configuration is performed as the following steps describe:

- In the `ENACTOR_HOME/conf` directory there must be a `crypto.properties` file that would specify the location of the user's keystore and the keystore password. User's key must be trusted by the services that the enactor is using.
- In the BREIN environment the enactor executes application services via the client side BREIN Gateway. To enable this, a `gateway.properties` file is required in the `ENACTOR_HOME/conf` directory. This file should contain the following line

```
gateway.address=http://gateway.host/path.to.gateway.binding
```

- The user's keystore should also have the public key of the service Gateway if it is used.
- In the `ENACTOR_HOME/conf` directory there should be the `cxfl.xml` file. Users should edit the following fields in this file:

```
<cxfl:outInterceptors>
  <entry key="signaturePropFile" value="crypto.properties"/>
  <entry key="encryptionUser" value="gateway alias"/>
</cxfl:outInterceptors>
```

```

        <entry key="user" value="user alias"/>
    (...)
    <cxfl:inInterceptors>
        <entry key="signaturePropFile" value="crypto.properties"/>
        <entry key="user" value="gateway alias"/>
        <entry key="decryptionUser" value="user alias"/>
    </cxfl:inInterceptors>

```

Components that act as adaptors between the enactor interfaces and the evaluation services can be plugged in to the enactor. Plug-ins can be put in the `./conf/plugins` directory. Jar archives in that directory will be added into the classpath and loaded by the class loader. The enactor additionally requires the location and role of the plug-ins. This can be done in the `plugins.xml` file:

```

<?xml version="1.0"?>
<components>
    <component>
        <class>
com.elsagdatamat.vim.servicediscoveryregistryplugin.SDRProcessDiscoverer</class>
        <role>Discoverer</role>
    </component>
    <component>
        <class>
com.elsagdatamat.vim.sessplugin.SESSProcessSelector</class>
        <role>Selector</role>
    </component>
    <component>
        <class>
com.elsagdatamat.vim.slanegotiatorplugin.SLANegotiatorProcessNegotiator</class>
        <role>Negotiator</role>
    </component>
</components>

```

There are two user interface implementations to drive the enactor: the Workflow Designer (the GUI) and the simple Command Line Interface (CLI).

In the binary distribution of the `simpleCLI` type:

```
./simplecli
```

The output is:

```

Workflow Enactment Engine Version 2.1
(c) Copyright 2008, University of Southampton IT Innovation Centre,
    Grid Systems, S.A. and Elsasg Datamat

usage: java uk.ac.soton.itinnovation.cli.SimpleCLI [option value]
{workflow-URI | workflow-filepath}
-i      Input Data (String). inputParameter=Value pairs. For more than
one

```

```
inputs use additional -i parameters.
-g set debug messaging on
-h Shows this message
-q Enable QoE monitoring
-v toggle on off verbose mode and logging of execution
```

As an example the execution of a BreinGrounding follows

```
./simplecli resources/gateway_add.xml -v -g
```

```
Workflow Enactment Engine      Version 2.1
(c) Copyright 2008, University of Southampton IT Innovation Centre,
Grid Systems, S.A. and Elsasg Datamat

Give http://tempuri.org//GatewayMock_add/inputs/operandA 1
Give http://tempuri.org//GatewayMock_add/inputs/operandB 2
Enactment is starting
Looking for listeners for urn:uuid:d310e0bf-78fc-4aa7-b442-
ce3e6a95b3b3

Process evaluated: http://tempuri.org//GatewayMock_add
Process started: http://tempuri.org//GatewayMock_add
Invoking service...
Output is 3
```

A `client.log` file is generated in the execution directory. The log file contains all necessary information to debug the enactment.

### **Semantically Enhanced Service Selector**

For this component details, please refer to the Service Broker toolkit Manual.

### **SLA Translator**

For this component details, please refer to the Service Broker toolkit Manual.

### **Service Discovery Registry**

For this component details, please refer to the Service Broker toolkit Manual.

### **SATSLA GUI**

For this component details, please refer to the Service Publishing and Registry toolkit Manual.

### **SATSLA Repository**

For this component details, please refer to the Service Publishing and Registry toolkit Manual.

### B.1.3 Usage Instructions

The basic aim of the Workflow Management Framework is the modelling enactment and concretization of abstract workflows. Basically, it is composed of all the components related to workflow modelling and enacting, plus those required for service publishing, discovery and selection.

The Workflow Management Framework involves the Workflow Designer, the Workflow Enactor and all the components coming from the Service Broker and the Service Publishing toolkits. So, from the Service Broker toolkit the components are the Service Discovery Registry (SDR), the Semantically Enhanced Service Selector (SESS), and the SLA Translator. From the Service Publishing toolkit the components are the SATSLA Repository, the SLA Translator, the SATSLA GUI, and the Service Discovery Registry GUI (SDR GUI). The relationships between the components are shown in Figure 49.

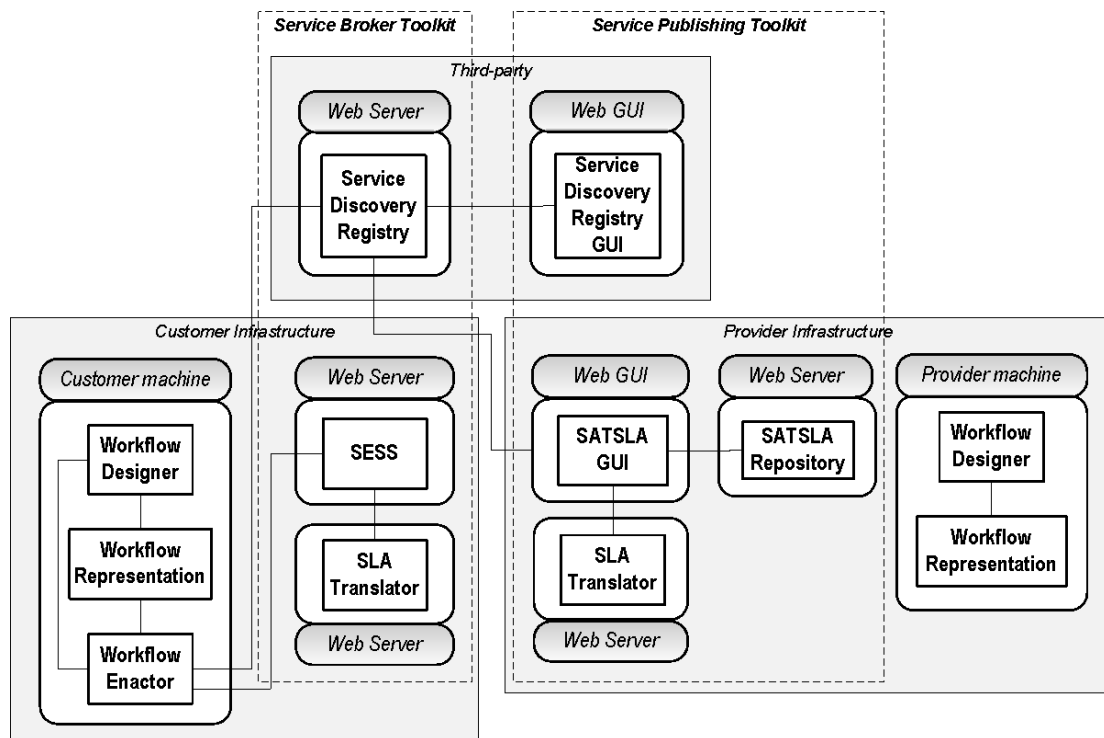


Figure 49: Workflow Management Framework Components

The process of publishing services, creating and enacting a workflow consists of a sequence of actions. Some of them are performed by the Service Providers, some others by the Customer.

Service Provider and Customer interact with the framework components generally as follows:

- 1.The Service Provider describes the service profile through the Workflow Designer;
- 2.The Service Provider uses the Service Discovery Registry GUI to register the service profile;
- 3.The Service Provider describes the SLA templates through the SATSLA GUI, registers the templates, and associates the templates to the already registered service (done at step 1);

4.The Customer takes advantage of the Workflow Designer to create the workflow to enact;

5.The Customer enacts the workflow.

Once the command to enact a workflow is issued, by the Workflow Enactor Command Line Interface (CLI) or through the Workflow Designer, the enactment process proceeds as follows:

6.The Workflow Enactor extracts the profile of a n abstract service from the workflow;

7.The Workflow Enactor searches for available services (able to concretise the abstract service) contacting the Service Discovery Registry;

8.The Workflow Enactor selects a service through the SESS, which does the selection considering the available SLA templates.

The interactions required with external components to complete the enactment process are as follows:

9.The Workflow Enactor negotiates the SLA contacting the SLA Negotiator (SLA Management Framework);

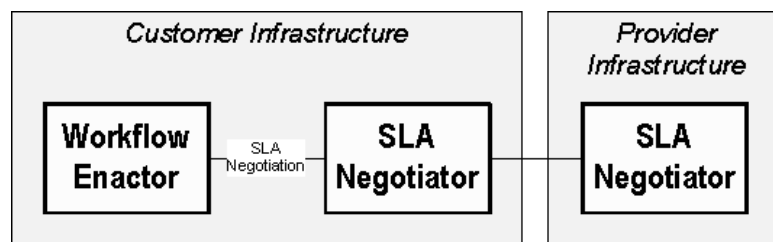


Figure 50: SLA Negotiation Interaction

10.The Workflow Enactor confirms the acceptance of the SLA (the contract) contacting the Contract Commitment Support (CCS) (Business Relationship Framework);

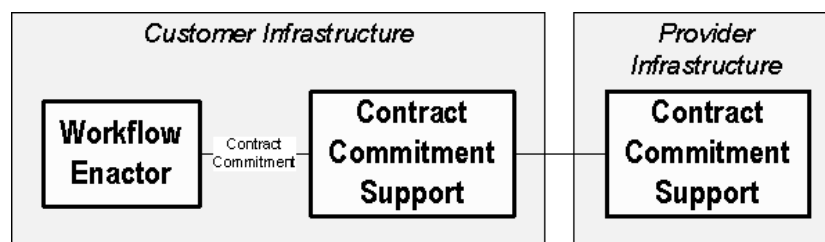


Figure 51: Contract Commitment Interaction

11.The Workflow Enactor invokes the service (eventually) through the customer / provider Gateways (Messaging Infrastructure Framework).

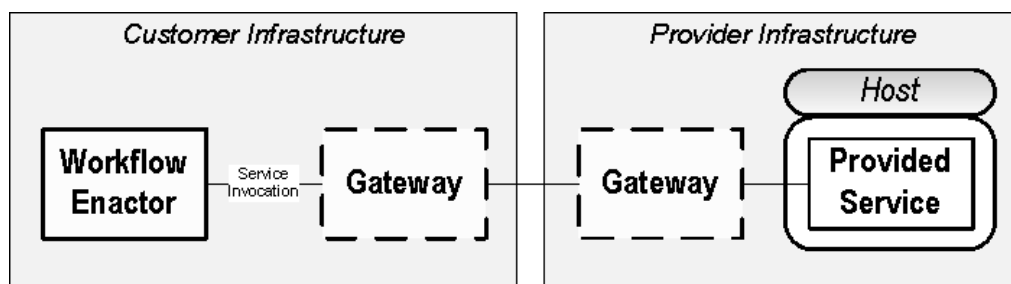


Figure 52: Service Invocation

An overall view of the Workflow Management Framework internal relationships and other relationships with external components is presented in Figure 53.

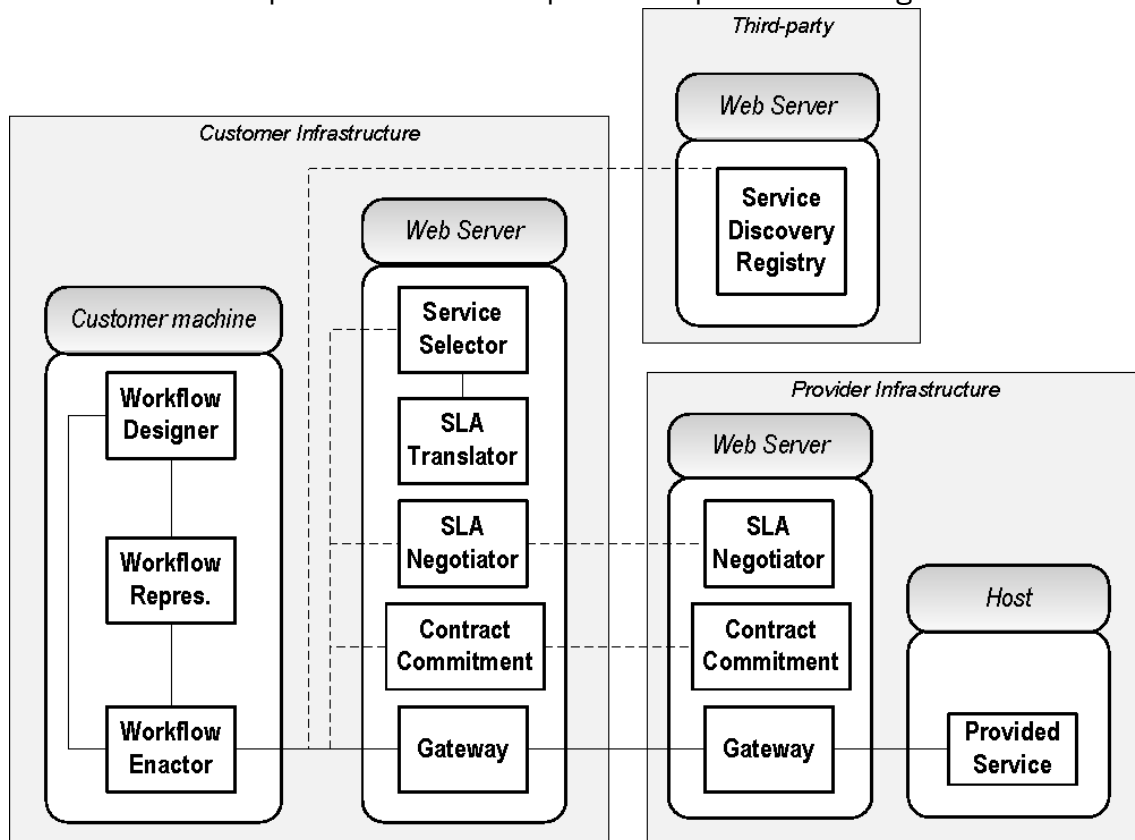


Figure 53: Workflow Management Framework Relationships Overall View

### Workflow Enactor

The basic aim of the Workflow Enactor is to dynamically evaluate and execute the application workflow. The application workflow may contain one or more abstract processes (nodes) – abstract processes do not contain execution endpoint and other required information - that need to be evaluated first.

### Workflow Evaluation

The enactor refers to components implementing its *ProcessDiscoverer*, *ProcessSelector*, *ProcessNegotiator* interfaces to help with workflow evaluation.

### Discovery Interface

The discovery interface searches for available processes that are able to perform the task that is described in the profile of the abstract process. It returns one or more candidate processes.

### Selection Interface

Selection interface designates the best candidate for an abstract process or for a workflow branch.

### Negotiation Interface

Negotiation is used to agree on a new SLA with the service provider that has been selected.

Workflow Evaluation procedure is shown in the following sequence diagram.

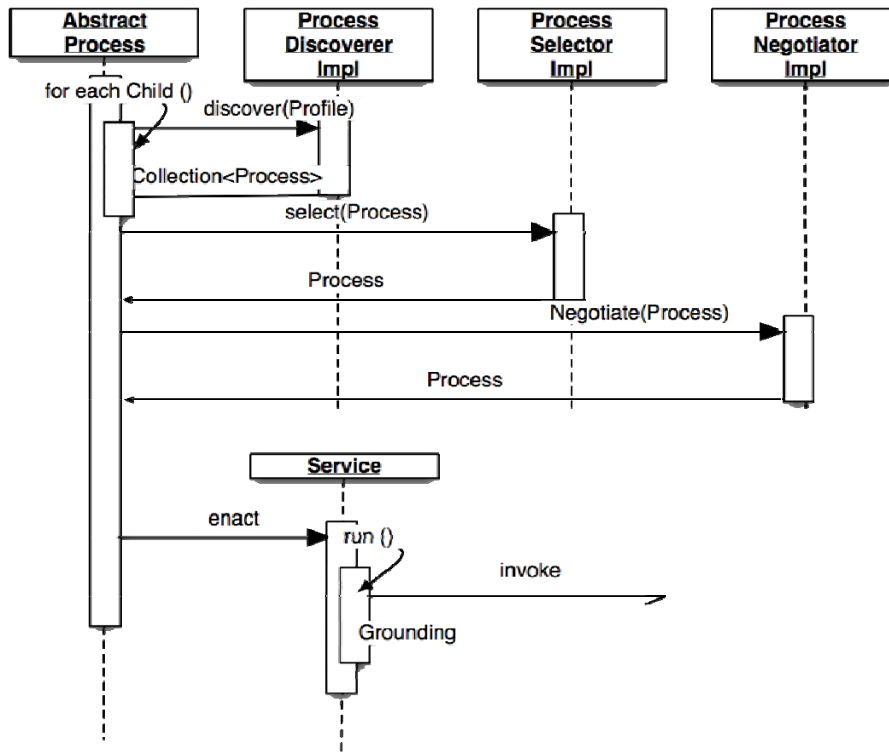


Figure 54: Workflow evaluation sequence and execution of a process

## Workflow Execution

The Workflow Enactor invokes the service if it has a concrete grounding description. Messages maybe forwarded to the service provider through the client-side gateway if gateway has been set up (see configuration).

## Workflow Enactor API

Apart from the Workflow Designer and the Workflow Enactor command line interface, a workflow can be enacted also taking advantage of the Workflow Enactor API. This API is provided by the PomService class. Most relevant interface functions are described in the following:

**Table 16 - Workflow Enactor API**

<b>Method</b>	void <b>enact()</b>
<b>Arguments</b>	None
<b>Return Value[s]</b>	None
<b>Description</b>	This function triggers the enactment process
<b>Method</b>	Object <b>getInput(String name)</b>
<b>Arguments</b>	String <i>name</i> – the name of the input
<b>Return Value[s]</b>	Object – the searched input
<b>Description</b>	This function returns the input identified by its name
<b>Method</b>	Object <b>getOutput(String name)</b>
<b>Arguments</b>	String <i>name</i> – the name of the output

<b>Return Value[s]</b>	Object – the searched output
<b>Description</b>	This function returns the output identified by its name
<b>Method</b>	<code>void setInput(String name, Object value)</code>
<b>Arguments</b>	<code>String name</code> – the name of the input
	<code>Object value</code> – the value of the input
<b>Return Value[s]</b>	None
<b>Description</b>	This function sets a process input providing name and value

### Semantically Enhanced Service Selector

For this component details, please refer to the Service Broker toolkit section.

### SLA Translator

For this component details, please refer to the Service Broker toolkit section.

### Service Discovery Registry

For this component details, please refer to the Service Broker toolkit section.

### SATSLA GUI

For this component details, please refer to the Service Publishing and Registry toolkit section.

### SATSLA Repository

For this component details, please refer to the Service Publishing and Registry toolkit section.