



xTier™ Workflow Service

The xTier™ Workflow Service is an ECA (event-condition-action) workflow engine. With no proprietary RDL (rule definition language), no interpreted scripting language and no additional IDE the xTier™ Workflow Service is a high-performance, scalable and easy to use workflow service.

Key Benefits:

- High-performance with Java or .NET rule implementations and no scripting language
- Increase application flexibility with XML based execution chain specification
- Increase developer productivity with no proprietary IDE – the developers work in their favorite IDE.

Introduction

The Workflow Management Coalition (WfMC) defines a workflow as an automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.

There are many workflow products in the marketplace that automate the definition and execution of business processes and most share some common attributes. Typically, most workflow engines impose a proprietary rule definition language (RDL) which has its own syntax and the ability to call methods written in some other well accepted language, such as Java, to represent this business process automation. In addition, this type of workflow system generally comes with its own IDE which additionally imposes a proprietary means of development and debugging. The main goal of this type of system is to enable business people to control the business workflow without any knowledge of the underlying implementation language. The disadvantage of this approach is that it splits the business logic into two pieces, half of which is written in a proprietary RDL and the other half implemented in a language such as Java. In order to debug this type of system, the developer has to somehow intuit where the actual implementation for a given bug or piece of logic is located (RDL or Java).

Current commercial workflow systems attempt to target a wide range of users. They provide visual tools for non-technical users and sometimes an open API for software developers. Although existing workflow systems may be adequate for end-users, they are not necessarily appropriate for developers. Some researchers [1] believe that the current common understanding of a workflow covers functionality that should probably be provided by separate components. Consequently, current workflow architectures are heavyweight and difficult to reuse and integrate with other environments.

All these problems make it hard for software developers to use existing workflow solutions whenever they require workflow functionality at the system (object) level, within their applications.

As a solution for these problems, Fitech Laboratories has developed a workflow service that we refer to as a “system-level workflow”. One of the main differences between the common understanding of “workflow” and our definition of “system-level workflow” stems from the scale of the process. Typical workflow products involve large-scale processes that execute on top of applications and business processes. In contrast, system-level workflow involves small-scale processes that execute within applications.

An important consequence of the scale difference relates to who the xTier™ Workflow Service targets. Non-technical users and software developers have very different requirements. Typical workflow products target process designers who are normally non-technical users. In contrast, the xTier™ Workflow Service targets technical people who develop and build applications (Developers and Architects).

Another consequence is related to the processing of workflow entities. In a typical workflow product, activities involve applications and humans. At a smaller scale, system-level workflow involves objects that make up applications. Having humans perform some workflow activities implies additional concerns that don't exist in a software-only context. Therefore, while a typical workflow product provides the functionality required to interact with people, the xTier™ Workflow Service employs a simpler mechanism that interacts only with other application parts. This approach separates the interface mechanism from the core workflow functionality, allowing them to be varied independently.

Designing Applications with the xTier™ Workflow Service

The xTier™ Workflow Service allows developers to split monolithic processes into smaller modular pieces and combine them freely using rules defined in XML. This, in turn, allows modifying application behavior without direct impact on the code defining a given rule, or the code that executes the execution set. It also provides the developer with the ability to reuse the small processing parts, defined in a given workflow, in new processes.

The xTier™ Workflow Service is based on the Event-Condition-Action (ECA) paradigm. The event component indicates when a rule has to be executed, the condition checks what action is to be executed next, and the action defines what needs to be done. In the xTier™ Workflow Service, an event is the result of rule execution, the condition checks the return value of the rule to determine which rule to execute next, and the action actually specifies which rule to execute. The xTier™ Workflow Service allows extracting the flow-independent actions and conditions and then freely combining them in XML to achieve the desired application functionality.

The main goal of the xTier™ Workflow Service is to provide developers with a convenient and easy to use API so they can utilize a customizable workflow when working on their daily projects. All rules are defined in XML and can be developed in a language favored by the developer (Java or .NET) using the IDE of their choice. Since the xTier™ Workflow Service does not impose a proprietary RDL, rules are executed by the Workflow Service with extremely minimal overhead. Additionally, the xTier Workflow Service provides configurable parse-time or run-time workflow validation to assist developers in ensuring that all input and output contracts within rules are met.

As with the other xTier™ services, the workflow service provides a common set of API's for Java and .NET. This common API allows for a single XML rule and execution set definition to be reused with both the Java and .Net rule implementations. This facilitates cross-paradigm workflows such that the successful execution of an execution set can be in either Java or .NET. Additionally, since the xTier™ workflow implementation is free of RDL it imposes no additional maintenance or development overhead and furthermore, provides extremely high-performance.

Increasing Application Flexibility

As a developer, you realize that requirements change as a natural part of the application development process. You can best address such changes by being prepared, in advance, by developing flexible and easily changed application architecture. Practically, however, this is a much more difficult endeavor than it sounds, particularly in light of decreased budgets and time frames. In many applications the “workflow” (or more simply, the order in which a process proceeds), experiences frequent changes. In a J2EE environment developers can address this issue by implementing various design patterns such as the “Dispatcher” pattern presented in the “J2EE Patterns Catalog”, however, this requires additional effort, the result is not guaranteed and most likely is not portable nor can it be reused. The xTier™ Workflow Service is a ready and reliable solution for this challenge providing a very flexible way of being prepared for requirements changes.

As with most of the xTier™ services, the workflow service uses Inversion of Control (IoC) for its configuration and rule specification. Programmers can concentrate on business logic implementing only simple classes which will be invoked by the workflow service during its work. The xTier™ Workflow Service will control (based on the XML workflow specification) the whole execution flow and will use the rule implementations (classes) when needed.

Rules

All workflow rules are defined in XML and can be easily reused within different workflow execution sets (execution chains). Every rule is completely stand-alone; i.e. a rule does not make any assumptions that some other rule was executed prior to it, or that some other rule will be executed after it. The state of the rule is managed by the Workflow Session. If a rule is not meant to be reused within other execution sets then it can be declared private to one execution set.

Every workflow rule can have multiple return codes that are either successes or failures. Note that return codes are strings that must be unique within a given rule definition.

A rule establishes a contract with the Workflow Service that certain input parameters will be provided to the Workflow Session before execution and certain output parameters will be added to the Workflow Session after execution. The output contract must be defined for each return code since different return codes may render different output parameters. The validity of input and output contracts are checked either at parse time or, if that is not possible, at runtime. Having input and output contracts allows the developer to freely reuse the same rule in different execution cycles as long as the contract is met. Note that some logic may require the determination of whether a given parameter is present in the session to occur at run-time. In such cases, input or output parameters are declared as optional by setting the optional attribute to true.

Every rule must specify at least one implementation for Java or .NET and may optionally have both Java and .NET implementations in the cases where the same rule definition is intended to be used in both platforms. Every implementation is composed of a class and method.

The following is an example of a rule definition taken from the workflow example provided with xTier™:

```
<rule name="validate.price">
  <ioc policy="new">
    <java class="com.fitechlabs.xtier.examples.services.workflow.WorkflowValidationRules$ValidatePriceRule">
      <!-- Constructor arguments. -->
      <ctor>
        <!-- Minimum price. -->
        <arg type="float64">0.01</arg>
        <!-- Maximum price. -->
        <arg type="float64">999999.0</arg>
      </ctor>
    </java>
  </ioc>

  <input>
    <param name="price"/>
  </input>

  <output>
    <success code="OK"/>

    <failure code="FAILURE">
      <param name="errMsg"/>
    </failure>
  </output>
</rule>
```

In this case, you will note that only a Java rule implementation is provided using IoC. There is a single input parameter – price. The output section defines the execution result, with the success code OK and the failure code FAILURE. The only output parameter specified is in the case of a failure and it contains an error message.

Rules can only be invoked from within execution sets and cannot be invoked directly from code. The following are requirements that should be met by all rule implementations:

- The method implementing a rule must be public. (As the method of interface.)
- The method's signature must have 1 parameter: WorkflowSession.
- The method must return a String containing predefined success/failure code(s).
- The method may optionally throw a WorkflowException or any of its subtypes (other exceptions will not be processed by the workflow service and will halt the execution chain in an unknown manner).
- The method cannot make use of class-level variables that may change throughout the course of execution.

The following is an example taken from the workflow example provided with xTier™.

```
public static class ValidatePriceRule implements WorkflowRuleBody {
    /** Minimum price. */
    private double min;

    /** Maximum price. */
    private double max;

    /**
     * ValidatePriceRule constructor.
     *
     * @param min
     * @param max
     */
    public ValidatePriceRule(double min, double max) {
        this.min = min;
        this.max = max;
    }

    /**
     * @see WorkflowRuleBody#invoke(WorkflowSession)
     */
    public String invoke(WorkflowSession session) throws WorkflowException {
        double price = session.getFloat64("price");

        if (price >= min && price <= max) {
            // Success return code.
            return "OK";
        }

        // Validation failed.
        session.addObject("errMsg", "Price must be within range [min=" + min + ", max=" + max +
            ", price=" + price + "]");

        // Failure return code.
        return "FAILURE";
    }
}
```

Actions

Actions are the third part of ECA (Event-Condition-Action) design principle. Actions can be of two types: execute and exit. An execute action tells the workflow engine to execute a certain rule. For execute actions you must specify a rule attribute, which is the rule that will be executed. An exit action forces the execution set to stop the flow of execution with a certain code. For exit actions you must specify a code attribute, which is the string representing the result of the execution.

Execution Sets

Execution sets are groups of rules within the same execution cycle. Rules can be freely reused across different execution sets.

Execution Sets follow the ECA (Event-Control-Action) paradigm:

Event – rule call returned some value.

Control – check the return value.

Action – based on the return value proceed to call another rule or terminate.

Execution set structure:

- Execution sets have one entry point and multiple terminal points. An entry point is the name of the rule or another execution set that gets executed first. A terminal point is the last rule called within some execution path of the execution set.
- Execution sets succeed if all the rules within the execution path succeed.
- Execution sets fail if one of the rules within execution path fails.
- Every call to a rule or another execution set specifies what action to take next. There are two types of actions: proceed and terminate. Proceed actions must specify which rule or execution set to call next. Terminate actions must specify an exit code. Note that unlike return codes in rules, exit codes are not successes or failures since execution sets succeed and fail differently from rules.
- Execution sets may have private rules defined within them. Private rules can only be used within the execution sets they are defined in. Private rules can be particularly useful when used to perform initialization of a Workflow Session at the beginning of execution path.

An execution set can be invoked either from code where you specify the first execution set to call or declaratively, via XML, from another execution set.

The following is an example of an execution set for placing a new order taken from the workflow example provided with xTier™:

```
<execset name="create.new.order" entry-rule="validate.price">
  <!--
    Creates an order object and adds it to the session.
    Note, that this rule is private to this execution set and
    cannot be accessible by other execution sets.
  -->
  <rule name="create.order.object">
    <ioc policy="new">
      <java class="com.fitechlabs.xtier.examples.services.workflow.WorkflowOrderCreationRule"/>
    </ioc>
  </rule>
  <input>
```

```
        <param name="qty"/>
        <param name="price"/>
    </input>

    <output>
        <success code="OK">
            <param name="order"/>
        </success>
    </output>
</rule>

<event rule="validate.price">
    <if code="OK">
        <action type="execute" rule="validate.qty"/>
    </if>

    <if code="FAILURE">
        <action type="exit" code="FAILURE"/>
    </if>
</event>

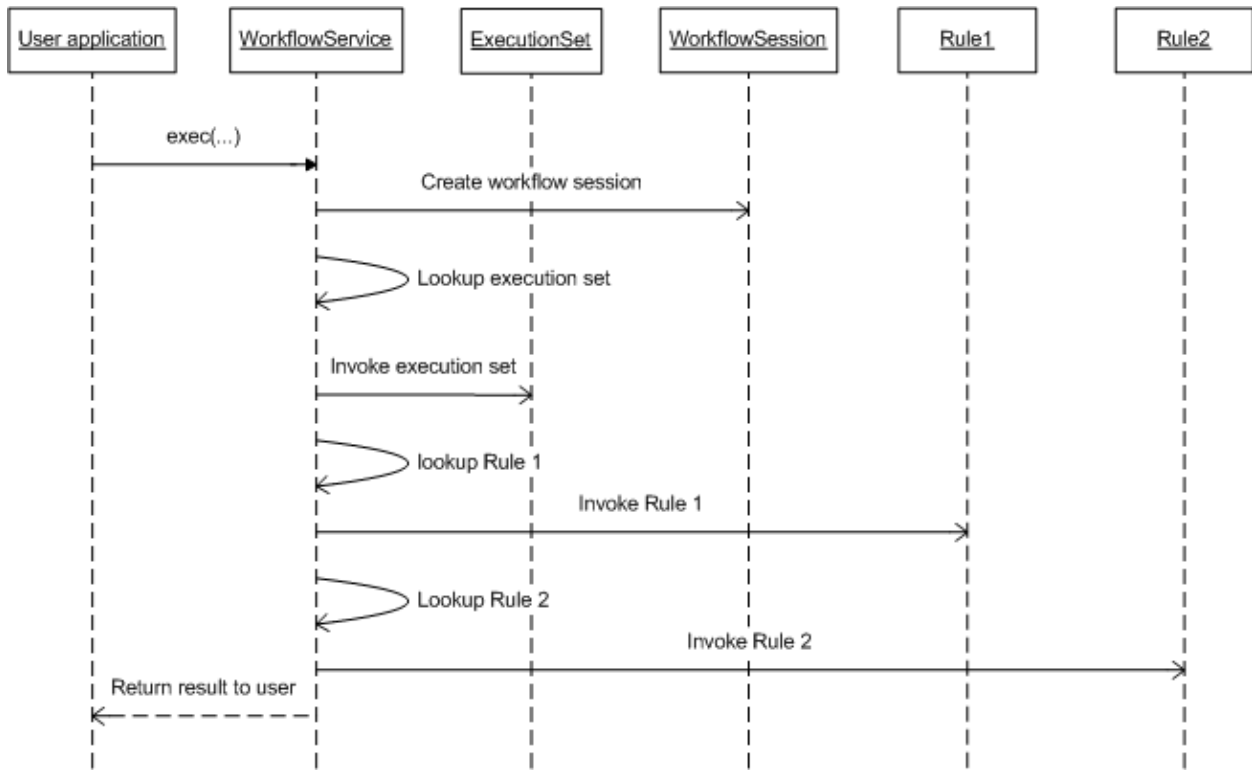
<event rule="validate.qty">
    <if code="OK">
        <action type="execute" rule="create.order.object"/>
    </if>

    <if code="FAILURE">
        <action type="exit" code="FAILURE"/>
    </if>
</event>

<event rule="create.order.object">
    <if code="OK">
        <action type="exit" code="OK"/>
    </if>
</event>
</execset>
```

There are three rules that can be called within this execution set. The first is the "create order object" rule which creates order object and if successful puts this object into the execution set context. The second is a proceed action and it validates the price. The third rule specified in this execution set is the "validate.qty" rule and it is a terminate action. Note the "exit" action type. In the event of an error the third error handling rule is called. The execution set specifies two input parameters, price and quantity.

The following diagram illustrates a workflow execution sequence.



The following is a code example taken from the workflow example provided with xTier™:

```
Map inputs = new HashMap();

// Store session inputs into the map. Note that these parameters are
// specified as inputs in the workflow XML configuration file.
inputs.put("qty", new Integer(100));
inputs.put("price", new Double(20));

// Execute execution-set 'create-new-order' from group 'order-processing'.
WorkflowResult result = workflow.exec("order.processing", "create.new.order" , inputs);

System.out.println("Workflow result: " + result);
```

Workflow Session

A Workflow Session is a data holder for all information required to be passed between the rules throughout the execution cycle. It is created before the first rule is executed and is passed into every rule call. Within a rule implementation the developer may retrieve data from the session or add data to it. Data retrieved from the session within the rule implementation should be declared as an input parameter in the rule and data added to the session should be declared as an output parameter of the rule.

Rules and Execution Sets are defined within an XML file. The following are some important details about the rule definition XML file that were not described above:

- Rule definitions are specified within regions (<region.../>) which can serve as logical groupings. As for any services, a workflow definition file may have multiple regions defined within it.
- It is possible to reference one workflow definition file from another utilizing the <include .../> tag.
- All public rules must be defined before execution sets definitions.
- All private rules must be defined at the beginning of an execution set definition.

Validation

The workflow configuration element “validation” specifies the validation parameters for the workflow service. This element has two attributes for two types of workflow validation:

- run-time - enables/disables run-time validation. Runtime validation ensures that all input and output contracts specified in the XML rule definition are actually followed by the implementation. It is important to set run-time validation to true when debugging and turn it off in the production deployment since it may adversely affect performance.
- parse-time - enables/disables parse-time validation. Parse time validation ensures that execution set is valid with respect to all input and output parameters specified in the rule’s XML definitions. Parse-time validation also checks for cycles in execution paths, verifies that rule and execution set names are not mistyped, checks that all returned codes returned by a rule are handled within the calling execution set, warns about unreachable events and verifies that all execution sets have at least one terminal event. It is recommended that parse-time validation is always turned on since it has no effect on performance.

References: [1] C. Bussler. *Enterprise-wide workflow management*. IEEE Concurrency, pages 32-43, July-September 1999.

Fitech Laboratories Inc.

Corporate Headquarters

300 Montgomery St., Suite 621
San Francisco, CA 94104
USA

Phone: 1-415-371-8234
Fax: 1-415-371-8237

East Coast Sales Office

330 Madison Ave., 9th floor
New York, NY 10017
USA

Phone: 1-646-495-5076

Fitech Laboratories Japan

Toranomon40 MT Bldg. 3F
5-13-1 Toranomon Minato-ku
Tokyo 105-0001, Japan

Phone: +81-3-5402-7711
Web: www.fitechlabs.co.jp