Levi – The Native BPMN 2.0 Execution Engine

A Workflow Engine based on Apache ODE's JACOB Framework

E. Sooriyabandara,
I. Jayawardena, K. Gallaba,
U. Pavalanathan, V. Nanayakkara
Department of Computer Science
and Engineering,
University of Moratuwa.
Moratuwa, Sri Lanka

M. Pathirage
Indiana University
Bloomington, Indiana, USA

S. Perera
WSO2 Inc.
Mountain View, CA, USA

Abstract— **In today's enterprise world, as more and more importance is placed on process automation and IT based governance, organizations tend to model and manage their business processes to achieve increased efficiency and productivity. The proper use of process modeling concepts in business scenarios enables designers to specify process requirements in terms of interactions enacted by human agents. Although Business Process Modeling is possible with languages like Business Process Execution Language (BPEL), they use more of a programing oriented view as oppose to human oriented view. Standardization of the Business Process Model and Notation version 2.0 (BPMN 2.0) provide a way to support inter-operation of business processes at human user level, rather than at the software engine. Although BPMN has being standardized, its wide adoption is limited by the lack of runtimes supporting BPMN. Although there are several BPMN implementations, they convert the BPMN to BPEL or another intermediate representation, which will yield mix results. In this paper, we discuss the design of Levi, a BPMN 2.0 runtime build using the underline constructs of ODE (Orchestration Director Engine), Apache based open source process engine. Unlike most other approaches, Levi supports BPMN natively using a concurrent runtime that supports Join pattern.**

Keywords- Business process; business process modeling; BPMN 2.0; business process execution; workflow engine; business-IT gap Introduction

I. INTRODUCTION

In today's enterprise world, as more and more importance is placed on process automation and IT based governance, organizations tend to model and manage their business processes to achieve increased efficiency and productivity. The proper use of process modeling concepts in business scenarios enables designers to specify process requirements in terms of interactions enacted by human agents. Although Business Process Modeling is possible with languages like Business Process Execution Language (BPEL), they use more of a programming oriented view as oppose to human oriented view. Standardization of the Business Process Model and Notation version 2.0 (BPMN 2.0) provide a way to support inter-operation of business processes at human user level, rather than at the software engine. Although BPMN has being standardized, its wide adoption is limited by the lack of runtimes supporting BPMN. Although there are several BPMN implementations, they convert the BPMN to BPEL or another intermediate representation, which has yield mix results. In this paper, we discuss the design of Levi, a BPMN 2.0 runtime build using the underline constructs of ODE (Orchestration Director Engine), Apache based open source process engine. Unlike most other approaches, Levi supports BPMN natively using a concurrent runtime that supports Join pattern. Business Process Management (BPM) is a management approach focused on aligning all aspects of an organization with the requirements of its clients. A BPM system can be viewed as a type of Process-Aware Information System (PAIS), which helps an organization make greater profits by improving the way they do business [1]. The efficiency and productivity enhancement of BPM systems make those useful for any type of organization [2]. BPM primarily focuses on the comprehensive management and transformation of operations presented in the processes of an organization [3]. A typical organization would have deployed hundreds or thousands of processes most of which controls the main sources of their revenue. Therefore, these processes must be constantly examined and managed on an ongoing basis to assure that they remain as efficient and effective as possible [2]. The performance of these processes must be evaluated to ensure that they meet the organization's business targets, which are based on critical metrics that relate to customer needs and organizational requirements [3].

The concept of BPM has been growing since the last two decades. In 2006, Zur Muehlen introduced a Business Process Management life cycle [4] which can be used to improve the way a company conducts its business in the long and short term. BPM is the follow up to Business Process Re-engineering and before that Total Quality Management philosophy. Business Process Re-engineering is a radical and revolutionary approach to improve business process. The Total Quality Management is incremental, evolutionary and continuous in nature. Concisely, it can be described that BPM integrates Business Process Re-engineering and Total Quality Management by using re-engineering approach to improve business quality.

To manage business processes, they have to be modeled and documented. One of the essential parts of business process modeling is choosing the most suitable modeling approach. Among the existing graphical modeling notations prominent modeling approaches are, Petri Nets, UML Activity Diagrams, Role Activity Diagrams (RAD), Data Flow Diagrams (DFDs), State-Transition Diagrams (STDs) [4-6]. Notations like UML and DFDs are focused on the informational perspective of a process (information flow involved in a process) while notations like RADs and STDs are focused on the behavioral aspect (the behavior of the activities and the actors) of a

process. But none of these are complete solutions which mean that using a model from one perspective will have an opportunity cost of not using the others. In the recent past, there have been efforts in developing web service-based XML execution languages for BPM systems, such as Web Services Business Process Execution Language (WSBPEL/BPEL). But these languages, which were designed for software operations, were not meant for direct humans use. Therefore, only very experienced programmers could work with such languages. Business people who do the initial development, management and the monitoring of processes could not take the advantage of these languages. This business-IT gap in the current BPM software does not enable business users to easily model and execute business processes. The reason is the approaches used in building business process engines.

Since business people are more comfortable with visualizing business processes in a flow-chart format there is a human level of "inter-operability" or "portability" that is not addressed by XML execution languages such as WSBPEL. To address this, Business Process Model and Notation (BPMN) were standardized to yield the inter-operation of business processes at the human level, rather than at the software engine level. The first goal of BPMN is to provide a notation that is readily understandable by all business users from the business analysts who create the initial drafts of the processes to the technical developers responsible for implementing the technology that will perform those processes [7].

BPMN provides a standard visualization mechanism for business processes defined in an execution optimized business process language. Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation [8]. An ontological analysis of BPMN 1.0 confirms the relatively high maturity of BPMN and identifies few potential shortcomings which are improved in future versions [9]. BPMN enables businesses to model their internal business procedures in a graphical notation and communicate these procedures in a standard manner. It follows the tradition of flowcharting notations for readability and flexibility. The Object Management Group (OMG) is using the experience of the business process notations that have preceded BPMN to create the next generation notation that combines readability, flexibility and expandability. BPMN advances the capabilities of traditional business process notations by inherently handling Business-to-Business (B2B) business process concepts, such as public and private processes and choreographies, as well as advanced modeling concepts, such as exception handling, transactions, and compensation [10-11].

BPMN 2.0 is a step forward for the whole business process management community because it introduces not only a standard graphical notation, but also concise execution semantics for process execution that can be used to enable the real execution of business processes that are modeled using it [11]. BPMN 2.0 provides a commonly agreed upon formal execution semantics by introducing concise execution semantics, thus overcoming the major drawback in the earlier versions such as BPMN 1.2 [1]. In addition to that BPMN 2.0 provides a notation and a model for business processes and an interchange format that can be used to exchange BPMN process definitions between different tools. Diagram interchange format facilitates the exchange of diagrams where as XML schema interchange allows easy sharing of model and its attributes. The goal of BPMN 2.0 is to enable portability of process definitions, so that users can take process definitions created in one vendor's environment and use them in another vendor's environment.

By providing a visual modeling language for business processes, BPMN 2.0 enables non-IT experts to communicate and mutually understand their business models. This progress in the area of business process management has resulted in widespread use of BPMN 2.0 as a modeling language [12].

This paper presents Levi, a highly concurrent BPMN 2.0 compatible process runtime. Although BPMN has achieve reasonable popularity, BPMN yet does not have wide runtime support.

Although there are several BPMN implementations, they convert the BPMN to BPEL or another intermediate representation, which has yield mix results. In this paper, we discuss the design of Levi, a BPMN 2.0 runtime build using the underline constructs of ODE, Apache based opensources process engine. Unlike most other approaches, Levi supports BPMN natively using a concurrent runtime that supports Join pattern.

Apache ODE, which is among the most influential open source process engines, provides a BPEL based process execution runtime. Since concurrency and join pattern is one of the key considerations while building a processes execution framework, ODE defines a runtime called JACOB (Java Concurrent Objects), a highly concurrent implementation of join pattern, which support persistent executions. This layer is independent from BPEL, and Levi implements support for BPMN 2.0 on top of JACOB runtime. One of the main challenges of building Levi was mapping BPMN constructs to underline JACOB runtime. We will discuss challenges, design, and solutions we encountered while building Levi, and critically analyze its effectiveness.

Section 2 of this paper describes the existing approaches of implementing the BPMN runtime and the merits and demerits of mapping BPMN 2.0 into different intermediate exchange formats. The reasons for building Levi is explained in section 3. Section 4 discusses the design of Levi and section 5 explains the implementation of BPMN 2.0 runtime in Levi. Section VI describes the outcome of the work and we explain the future work in section VII.

II. Related Work

The effort of building BPMN 2.0 execution engines has started since the initial release of the BPMN 2.0 beta specification in August 2009. Many vendors considered BPMN as a visual notation to BPEL and started creating BPMN 2.0 execution engines that runs the processes in their existing BPEL engines. Consequently, they tried to map BPMN 2.0 semantics to BPEL semantics which is not straight forward, as we shall discuss this later in this section. Some other vendors used other intermediate exchange formats such as jPDL and XPDL to convert the BPMN 2.0 processes and then execute in their engines that does not support BPMN 2.0 process execution natively. At present, with the release of the final version of BPMN 2.0 specification in January 2011, there are several BPMN 2.0 implementers [13]. But almost all of them convert BPMN 2.0 processes into some intermediate form even though they claim native execution. Next sections highlight merits and demerits of mapping BPMN 2.0 to various intermediate exchange formats.

A. BPMN Runtime through BPEL

Most of the current BPMN 2.0 engine vendors use a BPMN 2.0 to BPEL mapping, which enables user to first model business processes using BPMN 2.0 constructs. However, at the runtime those implementations convert the BPMN 2.0 business process into one or more BPEL processes, and execute them using a BPEL engine. The use of such mapping has created many debates among BPM experts. Implementers of the ActiveVOS BPM suite [14] argued that native execution of BPMN 2.0 processes is complex, and that it is simpler to map BPMN processes to BPEL[15].

However, several publications [16-19] have pointed out that the conceptual mismatch between BPMN 2.0 and BPEL, and discussed the pitfalls of mapping BPMN 2.0 into BPEL. When converting a language to a different language, it is required to measure the feasibility of doing that conversion. Mainly the conversion should minimize, if not avoid loss of semantic representation of information. That means the transition between languages should establish a high extent of matching of main representation capabilities between the two languages and a matching of control flow support. When converting BPMN 2.0 to BPEL, there exists a significant mismatch of domain representation capability and control flow support.

Domain representation mismatch occurs when there is construct deficit within languages, which inhibits from stating certain domain aspects. This means that when a more expressive modeling language is converted into a less expressive modeling language, the translation will be at the cost of losing expressive power and thus, semantic information about the represented domain. BPMN 2.0 is more expressive than BPEL and hence the conversion will result in loss of details. There are a number of potential domain representation capability mismatches like state, events and system mismatches in these two languages. State mismatch occurs because BPMN has more expressive power than BPEL. i.e. BPMN keeps more properties of a process than BPEL do. When translating BPMN to BPEL, these additional properties will be neglected. Event mismatch occurs because BPMN has more event subtypes. i.e. several event types of BPMN maps to one BPEL event. When translating BPMN to BPEL it is required to provide additional information to convert BPMN events to BPEL events. System mismatch occurs due to the concepts like Pools and Lanes in BPMN which are not in BPEL. When translating BPMN to BPEL it is required to pay more attention to the semantics of BPMN Pools and Lanes in order to describe the process in BPEL, which can be costly [18].

Control flow support mismatch occurs when different languages support different workflow patterns. When converting one language into another, these workflow patterns need to be considered. There are number of mismatches between BPMN 2.0 and BPEL with regards to the support for various control flow concepts, which cause problems when converting BPMN 2.0 to BPEL such as translating advance synchronization patterns, structural patterns, and multiple instances patterns which are present in BPMN 2.0 and not present in BPEL [18].

Limitations of this mapping have been discussed in academia in a comprehensive manner. Most of the researchers in this field support the argument that BPEL is inherently block oriented like a computer program, while BPMN is inherently graph oriented like a flowchart, even though there are minor confusions about the structure of BPEL and BPMN 2.0 [20-21]. As pointed out by Weidlich et al. [19] this structural incompatibility is the key reason for the pitfalls of the mapping. It further discusses about the reasons for the pitfalls of the mapping and the myth of a straight-forward mapping.

Beside these reasons, the BPMN 2.0 specification itself describes that only a small subset of the BPMN 2.0 constructs are isomorphic with BPEL and can be mapped to BPEL directly. The specification further says that not all BPMN 2.0 processes can be mapped to BPEL in a straightforward manner. Because BPMN allows the modeler to draw almost arbitrary graphs to model the controls flow, whereas in BPEL, there are certain restrictions such as control-flow being either block-structured or not containing cycles. The specification [15] essentially says in the “extended mapping” section that engine vendors are on their own, noting “in many cases there is no preferred single mapping of a particular block, but rather, multiple WS-BPEL patterns are possible to map that block to”. This contradicts with the argument that this mapping is simpler than native BPMN 2.0 execution.

Guo et al. [16] and Indulska et al. [17] argue for the need bi-directional transformation between BPMN 2.0 and BPEL for a complete such mapping and the limitations of achieving it. [17] It uses the Bunge-Wand-Weber representation model to analyze the representational capabilities of BPMN 2.0 and BPEL4WS, and on that basis, argues that the translation between BPMN and BPEL4WS is prone to difficulties due to inconsistent representational capabilities. They also claim that their work serves as a theoretical cornerstone on which the development of better mapping support for BPMN 2.0 and BPEL4WS can be based on.

B. BPMN Runtime through jPDL

Similar to the mapping of BPMN 2.0 to BPEL, some argue that BPMN 2.0 to jPDL mapping is suitable for BPMN 2.0 execution engines. jPDL [22] is the jBPM Process Definition Language (JPDL) for jBPM [23], a Business Process Management Suite from the JBoss community. Even though jBPM claims that it support BPMN 2.0 process execution natively, it internally converts the BPMN 2.0 process definition in to jPDL definitions before executing the business process in the existing engine. jBPM implements BPMN 2.0 process execution on top of the jBPM Process Virtual Machine (PVM), which was originally built for executing jPDL processes hence requires a conversion [24]. More over jPDL is not an industry wide standard; it is just the language used only in the jBPM suite and can only be used by it. Hence this conversion is far from being accepted as a standard for executing BPMN 2.0 processes [25].

C. BPMN Runtime through XPDL

Some vendors use XPDL (XML Process Definition Language) as the intermediate format to run BPMN 2.0 processes. XPDL [26] is designed to exchange the process definition, both the graphics and the semantics of a workflow business process, among different workflow products [27]. Hence this conversion does not result in native execution of BPMN 2.0 processes.

III. WHY LEVI?

With the introduction the operational semantics for BPMN 2.0, it is now possible to build an engine that directly supports BPMN 2.0 - without the intermediate step of generating BPEL. As explained by Leymann [12], no BPEL at all is required to execute process models specified in BPMN 2.0. Levi is designed to be a native BPMN 2.0 execution engine, which can

be used to execute business process models that conform to the BPMN 2.0 specification.

Implementing a workflow engine is tantalizing and yet a daunting task. There are many non-functional requirements like robustness, efficiency and scalability expected from an enterprise level workflow engine. Open Source WS-BPEL 2.0 implementation like Apache ODE [28] has mechanisms to ensure concurrency, durable continuation, reliability, and recovery. It uses a framework called JACOB, which is a practical combination of ideas from the actor model and process algebra approaches to concurrency and continuation. The implementation of the BPEL constructs is simplified by limiting itself to implementing the BPEL logic and not the infrastructure necessary to support it [29].

Without reinventing the wheel, as ODE's BPEL implementation relies on JACOB framework to implement the BPEL constructs, Levi uses JACOB to implement BPMN 2.0 constructs. Most importantly, it serves as a proof of concept for exploring the possibilities of using Apache ODE and JACOB to execute BPMN 2.0 processes consisting of core BPMN constructs.

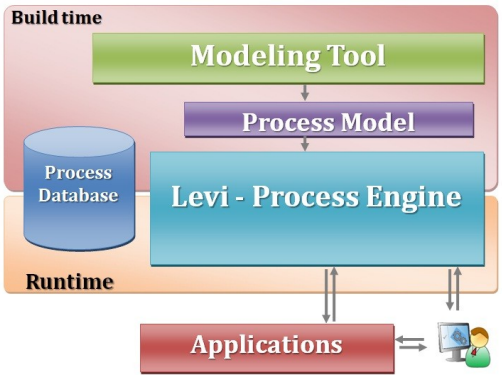

Figure 1: Major Building Blocks of the Levi Process Engine

IV. Architecture of Levi

A. Overall Architecture

Figure 1 shows the major building blocks of a BPMN execution engine. Users first describe their processes using BPMN and then deploy them in Levi, and it stores them in the process database. The process engine executes the process. Our discussion on Levi will focus on the execution engine that handles the runtime, compared to the build time of a BPMN model. This is due to the fact that for a given BPMN model, the build time occurs only once, whereas the runtime is expected to be functional each time that model is executed or managed/monitored through the administrator's or any other user's console.

1) Build time

This is when the user creates a BPMN model for a business scenario to fulfill his requirement. To model a BPMN process, a modeling tool such as BPMN2 Visual Editor for Eclipse [30] can be used. A typical modeling tool supports creating BPMN diagrams in a visual editor and generates the corresponding XML representation of that process model. After modeling the basic model in BPMN, the model must be made in to a process archive that can be deployed in Levi. To do this, additional artifacts such as the user input forms, WSDL files, process diagrams etc. must be bundled together with the created BPMN file. Once the process archive is deployed, it is stored in the Process Database of Levi.

2) BPMN Process Model

A BPMN process model is essentially an XML document that corresponds to the standard BPMN XML Schema document proposed by OMG. The Levi engine expects all the BPMN files to have a '.bpmn' extension and these BPMN files are validated when those are deployed to the system in the form of a business archive.

3) Format of a Process Archive

The process archive type identified by Levi is called the "Levi Process Archive" type, which is a zip archive renamed to have a *.lar* extension. A valid archive must have a single top most directory in which all the sub directories, BPMN files and other artifacts are included.

4) Runtime

This refers to two concepts both related to BPMN process execution, depending on the context where those are referred. The first concept is the actual execution time of a deployed business process within the execution engine. The other concept is the subsystem of the execution engine which handles the execution, management, and monitoring of deployed business processes. This is also referred as the backend of Levi. The frontend of Levi and/or a third party application (web/desktop/mobile) can connect to the backend as shown in the Fig. 1, and manage business process via a customized user interface.

B. Major Components of Levi

For better understanding of the architecture, Levi engine can be partitioned into four functional components: runtime service module, storage service module, user management module and utility module.

1) Runtime Service Module

The BPMN runtime of Levi is the component that handles the basic execution of the engine. It acts as the backend for the web user interface where the users interact to deploy, execute, and manage their business processes. Next section will discuss this in detail.

2) Storage Service Module

The StorageService implementation handles the persistence of process states and process variables in Levi engine. Process states and variables are persisted to the database whenever a new value is available or a value in the database gets modified. The execution engine retrieves the data from the database via the storage service module and uses it for further execution. The requirement to update the database to the latest state is due to the uncertainty of consistent communication between the backend database and the process engine. Especially when a process is paused or when an asynchronous task such as a user task get executed, the engine writes data to the storage expecting to retrieve them when the process resumes or when the asynchronous task ends. Also at the end of each task the engine does a storage update. In case of any kind of communication or server failures, the process engine can be restored to the latest running state by retrieving these process states and variables from the database. Since the StorageService component is a crucial part of the process engine, a considerable effort was required to build a standard storage service component.

3) User Management Module

The purpose of the user management module is to represent the concept of users (employees) and groups (departments) in a typical business environment. Groups are given different access levels and according to that members of that group can claim and complete business tasks.

The purpose of the user management component is to represent the concept of employees and departments in a typical real world business environment. A real world business process consists of different business tasks. These tasks can be divided into two major categories– tasks performed by human users and tasks done by machines/automated systems. In the BPMN 2.0 world, tasks done by human users are named as User Tasks. Hence user management component handles the non-execution part of the user tasks; execution part is handled by the runtime component.

4) Utility Module

Utility module consists of utility features such as process visualization and web form generation using template engine.

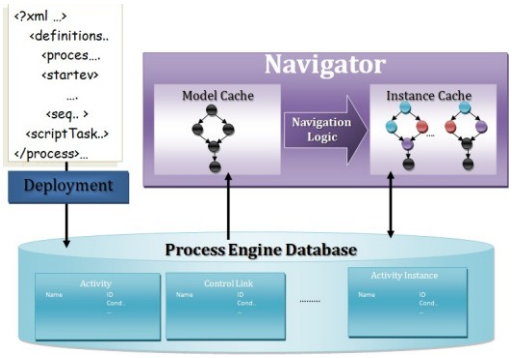

Figure 2: Deployment and Execution Architecture

C. Deployment and Execution Architecture

Figure 2 shows the high-level deployment and execution architecture of the business process execution engine Levi. When a business process archive is given as the input, a 'deployment' is created out of it. 'Deployment' is the runtime representation of the business process definition contained in the business archive. These process definition details and representation are stored in the Process Engine Database when the business archive is deployed.

There are two concepts to be clarified at this point – process deployment and process instance. A process deployment is a runtime representation of a business process, bundled in a *.lar* file. It is connected with the concept of process definition. Once a *.lar* file is deployed into the engine, only this process deployment is created. When a user wants to execute the business operations in that process, a process instance is created using the object model of the process definition. There can be multiples of process instances created from a single process deployment.

When a user wants to execute a business process, the process definition and object model of that particular process deployment is retrieved from the process engine database and a process instance is created in the runtime, as shown in figure 2. Properties of the process instance will be persisted in the database. When executing the process, the engine navigates through each BPMN 2.0 element in the process instance, until it reaches the end event. Process instance states are persisted in the database and retrieved when required.

D. Building Applications using Levi

The architecture of Levi engine is designed in such a way that real world business applications can be built on top of it with minimum effort. The major building blocks of the engine such as the RuntimeService, StorageService and UserManagmentService are exposed as APIs (Application Program Interfaces), which enables users of the engine to build a customized front end layer according to their business needs. This enables users to build different applications with less effort.

V. Implementation of BPMN Runtime

The BPMN runtime component handles the execution of BPMN logic within the Levi engine. It acts as the backend for the web user interface where the users interact to deploy, execute, and manage their business processes. The runtime is mainly composed of the runtime abstraction of a BPMN process; the ProcessInstance class, and the data types that represent the set of BPMN 2.0 constructs currently supported by Levi. All these types derive from a single type, called BPMNJacobRunnable and this class, in turn, derives from the JacobRunnable class of Apache ODE. This type hierarchy makes it possible to execute the Levi's representation of BPMN constructs and the process instances on the JacobVPU. Further, XMLBeans was used as the data binding tool to generate Java types from the XML representation of BPMN constructs and these types were used to bring in the definition of elements of the input BPMN documents to the context of the runtime. Each of the BPMN construct types acts as a wrapper for the corresponding XMLBeans generated type. Currently, Levi supports all of the simple BPMN 2.0 constructs as well as UserTask, SendTask and ServiceTask from the descriptive category as shown in figure 3.

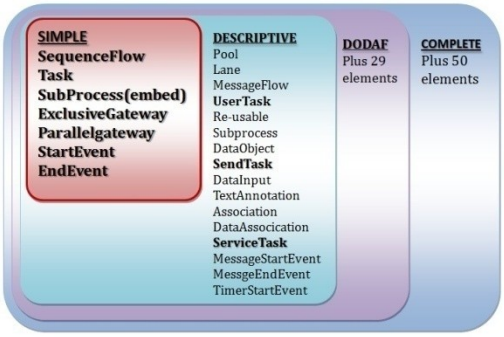

Figure 3: BPMN 2.0 Constructs

BPMNJacobRunnable defines some common methods related to all construct types and are used by the runtime. JacobRunnable defines an abstract method; run, which must be implemented by all of its derivatives. This method is executed by the JacobVPU and the construct related implementation is written in the run method of each construct type. For example, when implementing the ExclusiveGateway construct, the gateway related logic was written in its run method. Also it has a reference to an object of the type generated by XMLBeans; TExclusiveGateway. This instance brings in all the data present in the original XML element to the scope of the runtime.

Consider the following XML excerpt from a BPMN document which corresponds to an outgoing sequence flow

from an exclusive gateway element. This sequence flow contains the condition upon which is satisfied, the flow takes the path by referring to its target reference. The Levi's implementation of ExclusiveGateway can access the data such as the condition expression "i < 100000" only through the method getConditionExpression of instance of TSequenceFlow.

```
<sequenceFlow id="flow5"
              sourceRef="exclusiveGw2"
              targetRef="exclusiveGw1">
   <conditionExpression>
      <![CDATA[i < 100000]]>
   </conditionExpression>
</sequenceFlow>
```

Similarly, the implementation of ScriptTask accesses the script defined in the BPMN document's ScriptTask element by invoking the getScript method of the instance of TScriptTask. In the run method, it evaluates this script by using the context details of the current process instance, such as the process variables and the script type.

When a process is deployed to the engine in the form of a *.lar*, the runtime constructs the corresponding process definition by parsing and validating the BPMN document together with other dependent entities such as WSDL documents. The process definition (org.levi.engine.impl.bpmn.parser-.ProcessDefinition.java) is the static abstraction of a BPMN process inside the Levi engine. It is a data type that aggregates the BPMNJacobRunnable objects which correspond to the BPMN elements of the input BPMN document. When a process instance is created, it is passed with a reference to an instance of the corresponding process definition. The internal design of the process definition has been optimized for efficient navigation of BPMN elements to be used in constructing the process flow during the execution of the process instance.

At the initial stages of the design of the process definition, the iterator pattern was used to navigate the elements of a process instance. This decision was highly influenced by the linear arrangement of BPMN flow elements inside a BPMN document. This lead to incorrect runtime behavior when BPMN documents with elements arranged in a different order other than the order in which the process flow must occur were processed. From this, it was identified that the order of the elements of a BPMN document does not necessarily mimic the actual order of the process flow. BPMN uses an elegant solution to construct the process flow by using sequence flows and setting their source and target reference identifiers. Therefore, after considering all these factors, it was required to come up with a design which had the structure and characteristics similar to those of a graph which enables faster navigation compared to the linear iteration approach. As a solution, the previously described design was proposed in which the sequence flows are grouped into sets of sequence flows based on their target and the source reference IDs separately. These groupings are used as the major data structure in navigating the process elements by the runtime.

Execution of process instances includes starting, pausing, and stopping of process instances of deployed business processes. All these functions are executed when an authorized user gives corresponding command in the frontend. These commands are dispatched to the backend to be executed based on the process parameters. There are two types of executions. First type is executing multiple instances of a same process definition. In this, users can instantiate as many process instances as they wish from a given process definition and the engine is capable of isolating instances from one another and manage the execution. The second type is executing many process instances of different process definitions. The Levi engine supports these two types of process execution. The engine manages multiple instances of the different/same process definitions by resolving the relationship mappings among the users, tasks, process details and other process parameters of the instances accordingly.

VI. RESULTS

We have conducted a performance test for Levi with few process scenarios that use the script task construct. We used the following configuration for this purpose: Intel(R) Core(TM)2 Duo CPU T6670 2.20GHz processor with 2.00GB memory, on 32-bit Ubuntu 10.10 operating system.

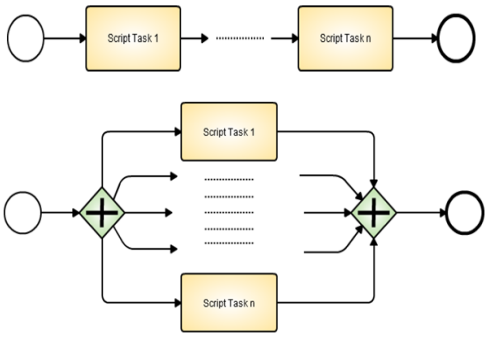

Figure 4: Sequential and Parallel Orientation of Constructs

Figure 4 shows the test scenario where the top process diagram shows sequential orientation of n ScriptTasks and bottom part shows the same process oriented in parallel. Figure 5 shows the comparison of running n ScriptTasks oriented sequential and parallel orientation.

As the graph suggests, the sequential approach involves more cost than the parallel approach. The running time of the parallel approach is almost lesser than that of the sequential approach. We get this behavior due to the difference of the parallelism involved in each approach. Parallel gateways are used to execute parallel tasks with the help of JacobVPU, in Levi engine. In the meantime for the parallel orientation, the running time is not constant since there is a considerable time delay involved when creating each ScriptTask construct.

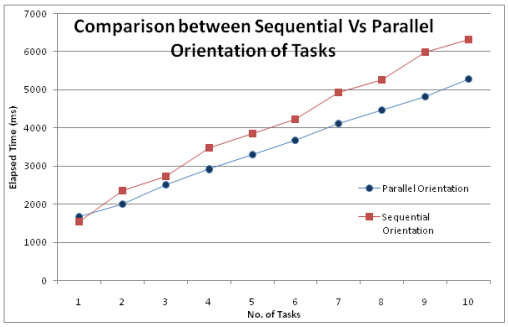

Figure 5: Comparison of Sequential Vs Parallel Orientation of Tasks

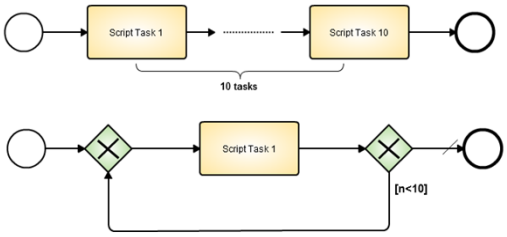

Figure 6: Sequential and Loop Orientation of Constructs

Figure 6 shows the test scenario where a business process of n ScriptTasks is oriented sequentially in the top process diagram and same process is modeled using a loop orientation in the bottom process diagram. Figure 7 shows the comparison of running time using the sequential and loop orientations as shown in figure 6.

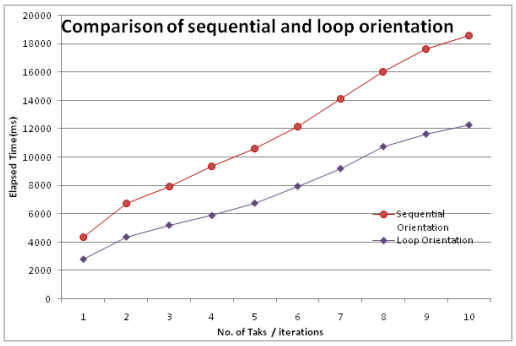

Figure 7: Comparison of Sequential Vs Loop Orientation of Tasks

According to the results, the loop model always takes less time than the sequential model. This is because creating a new object for each ScriptTask in the sequential orientation is an expensive operation to the JVM compared to evaluating conditional expressions in the loop orientation.

VII. Fututre Work

We were successful at implementing the basic BPMN 2.0 constructs in the Levi engine. Since we have designed our engine in such way that addition of new constructs are much simpler, and involves minimum amount of changes, any additional construct can be developed individually as a separate module and integrate to the runtime with less effort. We are working on expanding the set of supported standard BPMN constructs in future together with improvements to the implementations to the existing constructs. Further we are planning to add SOAP web services support for the Service Task and WS-HumanTask support for the User Task.

When we expose Levi engine as a product, performance is one of the most compelling factors to be successful among the competitors in the industry. A proper benchmark does not exist to test the performance of a BPMN engine. Therefore, creating a comprehensive benchmark for Levi is one of our major goals, which will also help to improve the industrial value of our project.

A BPMN2 Eclipse plugin [12] is under development to support the full BPMN 2.0 specification. The graphical editor can be integrated with Levi engine to design the processes and can be further improved to support deploying the designed processes through the IDE.

VIII. Conclusion

We have implemented Levi, a native BPMN 2.0 execution engine by using Apache ODE's JACOB framework. Levi is capable of deploying, persisting, navigating, and executing business processes claiming BPMN 2.0 execution conformance. It serves as a proof of concept for exploring the possibilities of using Apache ODE and JACOB framework to execute processes that consist of core BPMN 2.0 constructs. The implementation of BPMN 2.0 runtime in Levi proves that it is possible to build a BPMN 2.0 execution engine natively without converting into another intermediate representation such as BPEL or jPDL. It also contradicts the debate that converting BPMN 2.0 semantics into BPEL is the simpler way for building a BPMN 2.0 runtime. Further the native BPMN 2.0 runtime feature of Levi enables the rapid support for future expansion of BPMN 2.0 constructs set.

In this paper we have discussed the suitability of BPMN 2.0 to build a business process engine which fulfills both the requirements of business and IT people. The design and implementation of such an engine Levi is described in detail.

References

[1] P. Wohed, W. M. P. V. D. Aalst, M. Dumas, A. H. M. Hofstede, and N. Russell, "On the Suitability of BPMN for Business Process Modelling," in Proceedings of the 4th International Conference on Business Process Management, 2006.

[2] P. Harmon, Business process change: a guide for business managers and BPM and six sigma professionals, 2nd ed. Morgan Kaufmann, 2007, p. i-21.

[3] J. R. vom Brocke and Michael, Handbook on Business Process Management Volume 1, 1st ed. Springer, 2010.

[4] H. M. El-Bakry and N. Mastorakis, "Business process modeling languages for information system development," pp. 249-252, Aug. 2009.

[5] A. Arkin, Business Process Modeling Language. 2003.

[6] R. Cull and T. Eldabi, "A hybrid approach to workflow modelling," Journal of Enterprise Information Management, vol. 23, no. 3, pp. 268-281, 2010.

[7] B. M. Owen and J. Raj, BPMN and Business Process Management - Introduction to the New Business Process Modeling Standard. 2003.

[8] C. Badica and A. Badica, "Businss process modelling using role activity diagrams."

[9] J. Recker, M. Indulska, M. Rosemann, and P. Green, "Do Process Modelling Techniques Get Better ? A Comparative Ontological Analysis of BPMN," 16th Australasian Conference on Information Systems, 2005.

[10] T. Allweyer, BPMN 2.0 Introduction to the Standard for Business Process Modeling. 2010.

[11] Object Management Group, Business Process Model and Notation (BPMN), January. 2011.

[12] F. Leymann, "BPEL vs. BPMN 2.0: Should You Care?," in 2nd International Workshop on BPMN, 2010, pp. 8-13.

[13] "BPMN Supporters - Current Implementations Of BPMN." [Online]. Available: http://www.omg.org/bpmn/BPMN_Supporters.htm. [Accessed: 01-Oct-2011].

[14] "BPM, Business Process Management Software, Business Process Management Suite | ActiveVOS BPMS from Active Endpoints." [Online]. Available: http://www.activevos.com/. [Accessed: 01-Oct-2011].

[15] "BPMN or BPEL: which is simpler to understand? | VOSibilities." [Online]. Available: http://www.activevos.com/blog/bpel/bpmn-or-bpel-which-is-simpler/2009/11/19/. [Accessed: 01-Oct-2011].

[16] Y. Gao, BPMN - BPEL Transformation and Round Trip Engineering. 2008.

[17] M. Indulska, P. Green, J. Recker, and M. Rosemann, "Are we there yet? Seamless Mapping of BPMN to BPEL4WS," in 13th Americas Conference on Information Systems, 2007, pp. 1-11.

[18] J. Recker and J. Mendling, “On the Translation between BPMN and BPEL : Conceptual Mismatch between Process Modeling Languages,” in The 18th International Conference on Advanced Information Systems Engineering. Proceedings of Workshops and Doctoral Consortium, 2006, pp. 521-532.

[19] M. Weidlich, G. Decker, A. Großkopf, and M. Weske, “BPEL to BPMN : The Myth of a Straight-Forward Mapping,” in Proceedings of the OTM 2008 Confederated International Conferences.

[20] “BPMN vs BPEL: Are We Still Debating This? BPMS Watch.” [Online]. Available: http://www.brsilver.com/2009/11/19/bpmn-vs-bpel-are-we-still-debating-this/. [Accessed: 01-Oct-2011].

[21] M. Dumas and L. Garc, “Unraveling Unstructured Process Models,” in Proceedings of BPMN, 2010, pp. 1-7.

[22] “Chapter 18.jBPM Process Definition Language (JPDL).” [Online]. Available: http://docs.jboss.org/jbpm/v3/userguide/jpdl.html. [Accessed: 01-Oct-2011].

[23] “jBPM - JBoss Community.” [Online]. Available: http://www.jboss.org/jbpm. [Accessed: 01-Oct-2011].

[24] “jBPM BPMN | JBoss Community.” [Online]. Available: http://community.jboss.org/wiki/JBPMBPMN. [Accessed: 01-Oct-2011].

[25] “What’s in the Architecture?: XPDL,BEPL,JPDL,BPMNS,BPDM et al.. Standards and More Standards.” [Online]. Available: http://rabisblog.blogspot.com/2007/04/xpdlbepljpdlbpmnsbpdm-et-al-standards.html. [Accessed: 01-Oct-2011].

[26] “XPDL.” [Online]. Available: http://www.xpdl.org/. [Accessed: 01-Oct-2011].

[27] S. A. White and U. States, “XPDL and BPMN,” Management, pp. 221-238.

[28] “Apache ODE.” [Online]. Available: http://ode.apache.org/. [Accessed: 01-Oct-2011].

[29] “InfoQ: An Introduction to Apache ODE.” [Online]. Available: http://www.infoq.com/articles/paul-brown-ode. [Accessed: 01-Oct-2011].

[30] “Eclipse Modeling - BPMN2 Eclipse Plugin.” [Online]. Available: http://www.eclipse.org/modeling/mdt/?project=bpmn2. [Accessed: 01-Oct-2011].